École Polytechnique de l'Université de Tours
64, Avenue Jean Portalis
37200 TOURS, FRANCE
Tél. +33 (0)2 47 36 14 14
www.polytech.univ-tours.fr

Département Informatique
5e année
2013 - 2014

Rapport PFE

Mise en place d'un Serveur d'Intégration Continue et Qualité de Code

Encadrants
Nicolas Ragot
nicolas.monmarche@univ-tours.fr
Vincent T'Kindt
tkindt@univ-tours.fr

Étudiants
Anne CASTIER
anne.castier@etu.univ-tours.fr

DI5 2013 - 2014

Université François-Rabelais, Tours

Version du 3 mai 2014

Table des matières

Table des figures

Remerciements

Avant toute chose, je tiens à remercier Vincent T'Kindt, Nicolas Ragot et Pascal Meichel pour avoir encadré mon projet de fin d'études. Je tiens également à remercier le reste de l'équipe du service informatique, dont Mr Rousseau pour avoir résolu cet épineux problème d'Active Directory. Enfin et pas des moindres, j'adresse un grand merci à Léa Lehnebach et Raphaël Roger pour avoir servi de cobayes afin de tester le serveur d'Intégration Continue et de Qualité de Code.

Introduction

Dans le cadre de ma cinquième et dernière année à Polytech Tours au sein du département informatique, j'ai été amenée à réaliser le projet de fin d'études : "Mise en place d'un Serveur d'Intégration Continue et Qualité de Code", projet qui était proposé par Mr Vincent T'Kindt et Nicolas Ragot.

Le but de ce projet est de fournir un outil supplémentaire aux élèves développant d' importants projets informatiques (comme les PILs) dans le cadre de leurs études au sein de l' université François Rabelais. Le personnel de l' université, ainsi que les chercheurs, doctorants, seront également à même d' utiliser cet outil. Cet outil sera constitué d'un ensemble d' instruments et mécanismes permettant la mise en place de bonnes pratiques appelées "Intégration Continue" et "Qualité de Code".

Dans ce rapport, et étant donné la documentation considérable déjà produite pour ce projet, je m'attacherais à aller à l'essentiel. Pour cela, je détaillerais d'abord des résultats finaux et de l'aboutissement du projet en lui-même , avant de parler des méthodologies utilisées pour le développement de ce projet. Enfin, je parlerais du bilan que je dresse de cette aventure, sur le plan personnel et technique.

Notez au final, que l'ensemble des documents référencés dans ce rapport ont été mis en annexes à la fin de ce rapport, ce qui en explique la longueur.

Aboutissement et produits finaux

Dans cette première partie de ce rapport, je vais parler des résultats fournis -à savoir les machines virtuelles configurées ainsi que de la documentation créée pour ce projet, qui constituent l'aboutissement final de ce projet. J'évoquerais également les difficultés rencontrées lors de ce projet (certes peu nombreuses mais présentes quand même), avant de conclure sur les perspectives d'évolutions de ce projet.

3.1 Machines Virtuelles

3.1.1 Serveur

Bien évidemment, le résultat final le plus important de ce PFE est le Serveur d'Intégration Continue et Qualité de Code et est l'aboutissement de plusieurs mois de travail. Ce résultat n'est autre qu' une machine virtuelle opérationnelle, sous Windows 7, où tous les outils (Jenkins, SonarQube, Maven, Apache Tomat) décrits dans le cahier de spécifications ont été installés. Elle contient de plus toutes les bibliothèques de base de Visual Studio (nécéssaire pour faire tourner des projets construits sous Visual Studio avec Jenkins), grâce à l'installation de Visual Studio 2012 et 2010. Notez cependant, pour des raisons de sécurité, que seul un administrateur pourra accèder directement à la machine virtuelle. Les autres utilisateurs se contenteront d'accèder aux interfaces web des outils, et ceci afin de ne rien détraquer du serveur.

Vous pourrez trouver tous les détails relatifs à l'installation de ce serveur via le Manuel d'Installation Serveur ainsi que tous les détails relatifs à l'utilisation de ce serveur via le Manuel d'Utilisation. Je parle également des ces manuels dans ce rapport dans la partie suivante.

Je tiens également à revenir sur les tests effectués et sur les moyens de procéder. L'ensemble des tests s'est effectué de la manière suivante :

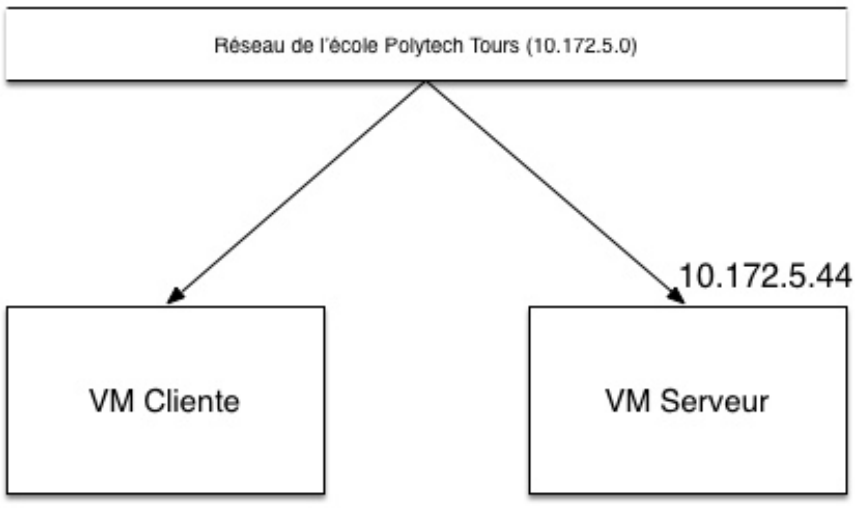

FIGURE 3.1 – Mise en place de tests

Le serveur d'intégration continue et qualité de code (à savoir la machine virtuelle installée et lancée sur un ordinateur physique de l'école) a été relié directement au réseau de l'école (après validation de Mr Meichel). Il possédait l'adresse ip 10.172.5.44. Une fois cette adresse connue, l'utilisateur final pouvait, depuis sa machine connectée au réseau de l'école par relais wifi d'une autre machine branchée sur le réseau de l'école, accéder au serveur et l'utiliser pour mettre en place ses projets (comme je l'ai décrit dans le Manuel Utilisateur). Cette expérience m'a permis de me mettre dans la peau d'un réel utilisateur et de contruire le Manuel Utilisateur de ce serveur.

3.1.2 Clientes

Pour faciliter la prise en main par les utilisateurs, j'ai pris l'initiative de construire plusieurs machines virtuelles cliente types, à-même d'utiliser le Serveur d'Intégration Continue et Qualité de Code et possédant déjà des outils de développement complet : Visual Studio, Eclipse, SVNTortoise (un client SVN afin d'utiliser facilement le serveur subversion du Redmine), Maven, SonarQubeRunner. Le fonctionnement de cet ensemble d'outils est d'ailleurs décrits dans le Cahier de Spécifications, et la manière de les utiliser dans le Manuel Utilisateur, je ne reviendrais donc pas dessus dans ce rapport.

Je tiens à insister sur le fait que ces machines sont extrêmement faciles à produire, si l'on considère la Manuel d'Installation Client, mais j'ai jugé utile d'avoir deux exemples à portée de main et directement utilisables (évitant la fastidieuse installation de Visual Studio pour les utilisateurs par exemple, ou des étapes de configuration rébarbatives (mise en place de variables d'environnement pour Maven par exemple)).

Les deux machines virtuelles clientes, complètes et entièrement fonctionnelles, crées pour ce projet à titre d'exemple, seront déployées sur le réseau de l'école afin d'être mise à disposition de tous les étudiants. Voici un bref descriptif des deux machines clientes produites :

Client Visual Studio 2010

Cette première machine virtuelle, construite sous windows 7, contient l'ensemble de logiciels et installations suivantes :

- Eclipse
- Visual Studio 2010
- Maven
- JDK 1.7
- TortoiseSVN
- SonarQubeRunner

Client Visual Studio 2012

Cette seconde machine virtuelle, construite sous windows 7, contient l'ensemble de logiciels et installations suivantes :

- Visual Studio 2012
- Maven
- JDK 1.7
- TortoiseSVN
- SonarQubeRunner

3.2 Documentations produites

Pour ce projet, il est évident qu'une large documentation devait être produite. En plus du cahier de spécifications obligatoires et du rapport final de ce PFE, j'ai décidé de produire une trilogie de manuels qui me semblait indispensable :

1. Manuel d'Installation Serveur
2. Manuel d'Installation Client
3. Manuel Utilisateur

L'ensemble de cette documentation va permettre à celui qui reprend le flambeau de ce projet de lui simplifier la vie et d'accélérer sa prise en main du projet.

3.2.1 Cahier de spécifications

Ce document, requis dans le cadre de n'imprte quel projet de fin d'études mené au sein de Polytech Tours, redéfinit l'ensemble des spécifications du projet :

Les besoins qui ont menés à ce projet (pour qui, pourquoi ce projet), sa problématique, comment l'élève va être amené à développer son projet de telle manière à répondre au cahier des charges, dans quels délais... C'est ce document qui a conclu la première phase menée pour ce projet, et qui a été finalisée par la soutenance de mi-parcours tenue à la mi-janvier en présence de mes deux encadrants Mr T'Kindt et Mr Ragot. C'est sur ce document que le reste du projet a été entièrement développé et pour lequel il était obligatoire de s'y référer en cas de moindre doute.

Dans mon cas, ce document a été accompagné d'un autre document "Veille Technologique" qui justifie pour ce projet l'ensemble des choix techniques réalisés pour ce projet. Il justifie par exemple, le choix de Jenkins plutôt qu'un autre serveur d'intégration continue, ou encore de tels ou tels plugins.

Cependant, il faut noter que ce document assez "lourd" (une bonne cinquantaine de pages), a été rendu légèrement obsolète sur certaines parties, modifiées suite à certaines découvertes imprévues (comme la découverte des fonctionnalités incomplètes du plugin cxx, par exemple). Ce genre de phénomène arrivant dans chaque projet, ce n'est pas un incident grave, mais je tenais à le souligner.

3.2.2 Manuel d'Installation Serveur

Ce document fait partie de la trilogie de documents (Manuel d'Installation Serveur, Manuel d'Installation Client, Manuel Utilisateur) qui va permettre à l'administrateur de prendre en main le serveur tel qu'il a été installé, et de l'utiliser depuis une machine client pré-configurée.

Grâce à de manuel, l'administrateur est à-même de réinstaller complètement le serveur d'Intégration Continue et Qualité de Code. Il contient ainsi le détail de l'installation des prérequis qui vont permettre de faire tourner (entre autres) Jenkins et SonarQube à savoir un JDK (Java Development Kit) récent, des installations Maven, Apache Tomcat... Il développe également l'installation de Jenkins et Sonar.

3.2.3 Manuel d'Installation Client

Ce document, plus court que son prédécesseur Manuel d'Installation Serveur, permet à l'administrateur de savoir quels sont les requis à installer et configurer pour une machine cliente afin que cette dernière puisse utiliser le serveur d'intégration continue.

Il peut également être mis à disposition des utilisateurs finaux (élèves...), puisqu'il ne contient pas de données confidentielles (celles indiquées sur le manuel ont été changées, comme le mot de passe sonar changé depuis l'interface web de sonarqube). Elle permet à n'importe qui d'utiliser le serveur depuis n'importe quelle plateforme (la plateforme décrite dans le manuel est Windows mais rien n'empêche des étudiants en informatique de l'adapter pour utiliser Jenkins et SonarQube depuis Ubuntu ou Mac OS X ...).

3.2.4 Manuel Utilisateur

Enfin ce dernier document, le plus copieux, est à destination de tous les utilisateurs du serveur, administrateur compris. Il décrit de quelle manière mettre en place un projet complet sous le serveur d'Intégration Continue et Qualité de Code et ceci dans l'ensemble des cas testés à savoir :

— Projet Java construit sous Eclipse, utilisant la technologie Maven
— Projet C# construit sous Visual Studio
— Projet C/C++ construit sous Visual Studio

Le but étant de permettre une prise en main rapide, on pourrait également redéfinir ce guide comme un guide "quick-start" étoffé et palliant de multiples cas. Bien sûr, il est laissé à l'utilisateur le choix de s'en inspirer pour implémenter ses propres projets dan des technologies non testées (il devra pour cela se rapprocher de l'administrateur actuel du serveur).

3.3 Difficultés rencontrées

Bien sûr, comme pour tous les projets, tout n'a pas toujours été rose et j'ai été confrontée à de nombreuses difficultés.

J'ai du gérer mon impatience à vouloir installer, configurer, tester le serveur... En d'autres mots, j'avais envie de commencer tout de suite par la partie "amusante" du projet et devoir commencer par une longue phase de recherches, de documentations, de justifications de mes choix n'a pas été facile à accepter pour ma part. Cependant je remercie mes encadrants de m'avoir poussée à travailler sur cette phase de recherches, tant elle a été importante par la suite et m'a facilitée la vie lors de la seconde partie de mon projet, lors de l'installation du serveur. Les bases saines et claires posées lors de la soutenance de mi-parcours (finalisant cette phase de recherche) avec Mr T'Kindt et Mr Ragot m'ont permis de savoir clairement ou j'allais et ce que je devais faire pour mener à bien l'installation de ce serveur.

Durant ce projet, j'ai également été confrontée à de nombreuses difficultés techniques. En effet, n'ayant pas d'autres connaissances autres que celles plus "théoriques" enseignées en cours, je me suis heurtée à de nombreux soucis techniques. Le manque de ressources, de documentations et d'explications claires sur Internet concernant l'ensemble des outils étudiés ainsi que ceux sélectionnés pour le serveur (Jenkins, SonarQube, les plugins rattachés à ces deux outils...) a clairement constitué l'une des difficultés majeures de ce PFE. Je ne compte plus les jours passés à tatônner pour trouver les raisons des erreurs et non-fonctionnement du serveur pour des cas "particuliers" (et j'entends par "particuliers" l'ensemble des projets qui ne sont pas réalisés en langage java/maven...). J'ai, par exemple, passé presque 3 semaines à essayer d'intégrer à Jenkins et SonarQube un projet plus avancé (un projet en C# développé par Léa Lehnebach sous l'encadrement de monsieur Romain Raveaux).

Ces difficultés, cependant, ne sont pas que dûes à mon manque de connaissances sur l'intégration continue mais sont également dûes à mes connaissances assez pauvres sur certains environnement de travail (Visual Studio par exemple). J'ai du ainsi prendre en main de nombreux projets différents, langages, envrionnements de travail pour ce projet, ce qui ne s'est pas non plus fait en deux jours.

De plus, j'ai dans ce projet été confrontée à quelques soucis de communications. En effet, avoir 3 encadrants n'est pas toujours évident pour transmettre les informations simultanément. Bien sûr, je transmettais des rapports chaque semaine par mail, mais lorsque j'avais besoin de prendre rapidement des décisions quant à la suite de mon projet, il me fallait réunir mes 3 encadrants dans des délais assez brefs afin de ne pas rester bloquée. Heureusement pour moi, Mr T'Kindt, Mr Meichel et Mr Ragot se sont montrés disponibles et présents tout au long du projet.

Enfin et pas des moindres, j'ai parfois du faire face à de très grandes déceptions. La dernière en date a été la découverte d'une "caractéristique" particulière du plugin cxx (supportant le langage C/C++) de SonarQube. Ce plugin ne lance pas l'analyse du code, il faut lui fournir avant... Autant dire que ce plugin s'est de suite rendu moins intéressant puisqu'il ne faisait pas le même travail que ses collègues (à savoir les plugins supportant le langage C# et Java). Après discussions avec mes encadrants, nous avons tout de même décidé de le garder, puisqu'il gardait d'autres caractéristiques assez importante d'un point de vue qualité de code (complexité des abaques par exemple).

Cependant je garde un bilan très positif de ce projet et je considère, j'espère à juste titre, que la plupart des difficultés présentées ci-dessus ont été surmontées ou contournées.

3.4 Perspectives d'évolutions

A ce jour, je peux assurer que le serveur, tel qu'il est fourni, est entièrement fonctionnel et les documentations qui l'accompagnent sont suffisamment complètes pour que celui qui s'en donne la peine soit à même de le reprendre en main et d'assurer son administration. Ce serveur, tel que je l'ai crée, respecte ainsi le cahier des charges fourni et le cahier de spécifications écrit durant la première phase de ce projet de fin d'études.

Cependant, il est de bon ton de reconnaître que ce serveur n'a pas été soumis, par manque de temps, à de vrais tests de la part de ses utilisateurs finaux. On ne peut à ce jour que supposer le bon fonctionnement du serveur lors d'une montée en charge, par exemple.

Ainsi, à mon sens ce projet peut encore être amené à évoluer sous l'impulsion d'une tierce personne, dans le cadre d'un PFE par exemple ou d'un moniteur engagé pour faire ce travail.
Cette personne pourrait ainsi redécouper ses tâches de la manière suivante :

1. Reprendre en main le projet : Cette première étape sera longue et déterminante et permettra, à la personne en charge de la suite de ce projet, de prendre en main et tester le serveur d'intégration continue et qualité de code, comprendre les enjeux qui s'y attachent, prendre connaissance de l'ensemble de la documentation déjà créée....
2. Apporter plus de polyvalence : L'ensemble des outils installés permet à d'autres langages (en plus du C/C++, C#, Java/Maven) d'être installés. Il serait ainsi sympathique de permettre une meilleur polyvalence du serveur en ajoutant et testant le support d'autres langages : PHP, Python, Ruby... ou en ajoutant la prise en charge de moteurs de productions pour le support de langages déjà testés : CMake, Nant...
3. Mettre en place une vraie période de tests : L'avantage de ce nouvel administrateur permettrait, lors de la mise en place des PILs pour l'année 2014-2015, d'accompagner ces PILs dans l'utilisation du serveur d'Intégration Continue et Qualité de Code et d'être sûr à 100% que ce serveur d'intégration continue et fonctionnel.

4. Si l'étape précédente s'est bien déroulée, alors l'administrateur pourra accompagner le transfert de ce serveur vers le parc informatique de l'école (où se situe déjà le serveur Redmine de l'école Polytech Tours).

Bien entendu, le projet étant déjà fonctionnel, on peut tout à fait le laisser tel quel. Cependant je pense que les étapes proposées ci-dessus ne seraient pas de trop. Qui plus est, cette suite ferait le bonheur d'un étudiant, qui pourrait avoir le temps nécéssaire pour se consacrer et se former à ces nouvelles technologies comme j'ai pu l'être durant mon projet de fin d'études. Ou laisser comme ça mais risque de mal utilisation, problème si ça tourne mal...

Méthodologies utilisées pour le développement du projet

Cette partie va permettre d'expliquer comment ce projet a été mené, les méthodologies utilisées, l'adaptation face aux contraintes diverses, ainsi que la planification du projet et le respect des délais.

4.1 Planification du projet

Il faut tout d'abord rappeler que le projet a été découpée en 3 grandes phases, fixées dès le début par Mr TKindt lors de notre première réunion et à laquelle je me suis tenue tout au long du projet :

1. Une première étape de prise en main des technologies utilisées à l'aide de ressources fournies par Mr T'Kindt. Cette première étape a mené ensuite à une phase de documentations, de recherches sur la conception future du serveur, phase que l'on a appelé "Veille Technologique". Nous avons pu clôturer cette première partie lors de la soutenance de mi-parcours tenue à la mi-janvier et qui a permis de redéfinir les spécificités du serveur, les outils... Et donc tout ce qui permettait de partir sur des directives claires pour installer correctement le serveur.

2. Une seconde étape a été ce que l'on a appelé l'étape d' "Intégration Réseau". En effet, une fois les bases posées, j'ai pu installer le serveur dans son ensemble, faire une série de tests basiques pour chaque cas particulier (Projet Java/Maven sous Eclipse, Projet C/C++ sous Visual Studio, Projet C# sous Visual Studio 2010 et 2012...). Je me suis également attelée à faire éxécuter mon projet par d'autres élèves de cinquième année, afin qu'ils puissent tester par eux-même mon serveur. Malheureusement, ce nombre d'élèves n'est pas très élevé par manque de temps sur la fin du projet, mais il faut noter que l'ensemble des retours reçus est resté très positif !

3. Enfin la dernière étape, l'une des plus importantes, a été menée en parallèle aux deux précédentes étapes : C'est l'étape de documentation. En effet pour ce projet, et comme vous avez pu le remarquer, une quantité considérable de documentations a été produites, et cela à travers plusieurs manuels : Manuel Utilisateur, Manuel d'Installation Client, Manuel d'Installation Serveur... Cette documentation était nécéssaire, puisqu'il fallait que le futur administrateur (cf partie Perspectives d'évolutions) soit a-même de reprendre en main le serveur et d'en comprendre ses enjeux.

Ce découpage, mis en place dès le départ Mr T'Kindt et moi-même, a donc permis un développement linéaire et méthodique du projet, qui s'est déroulé sans (grosses) encombres. On peut d'ailleurs y retrouver certaines composantes du développement "cycle en V" :

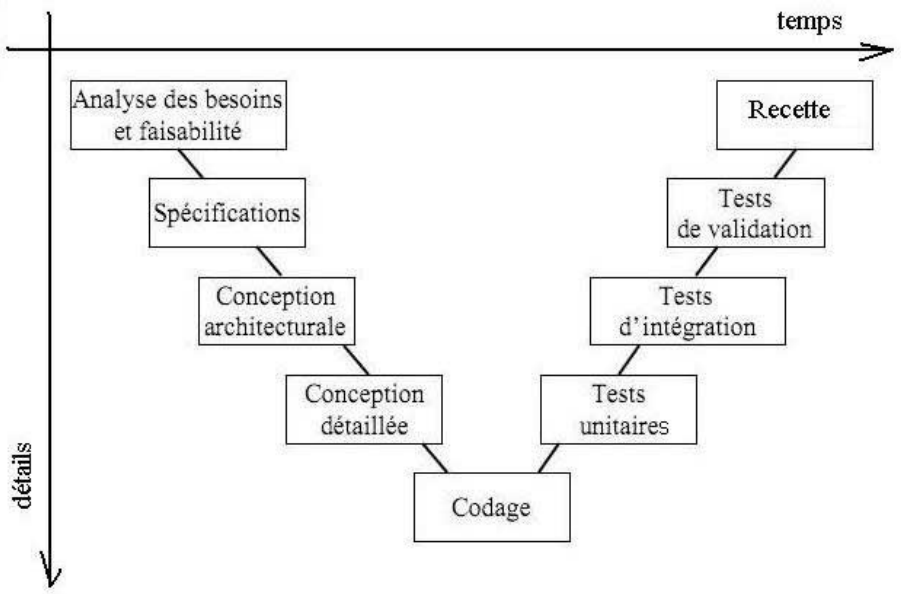

FIGURE 4.1 – Cycle en V ?

On voit bien que les étapes *"Analyse des besoins et faisabilité"*, *"Spécifications"*, *"Conception architecturale"*, *"Développement"*, *"Tests Unitaires"* et *"Tests d'intégration"* ont été réalisés dans ce même ordre. Cependant, plus que les étapes de réalisation du projet, il peut être intéressant de vérifier si délais et planning de développement (tenus dans le cahier de spécifications) ont été tenus.

4.1.1 Diagramme de Gantt

Revenons à nos propos tenus dans le cahier de spécifications, où nous montrions le planning de développement du projet suivant : Dans l'ensemble, le projet s'est déroulé comme ce qui avait été prédit, le

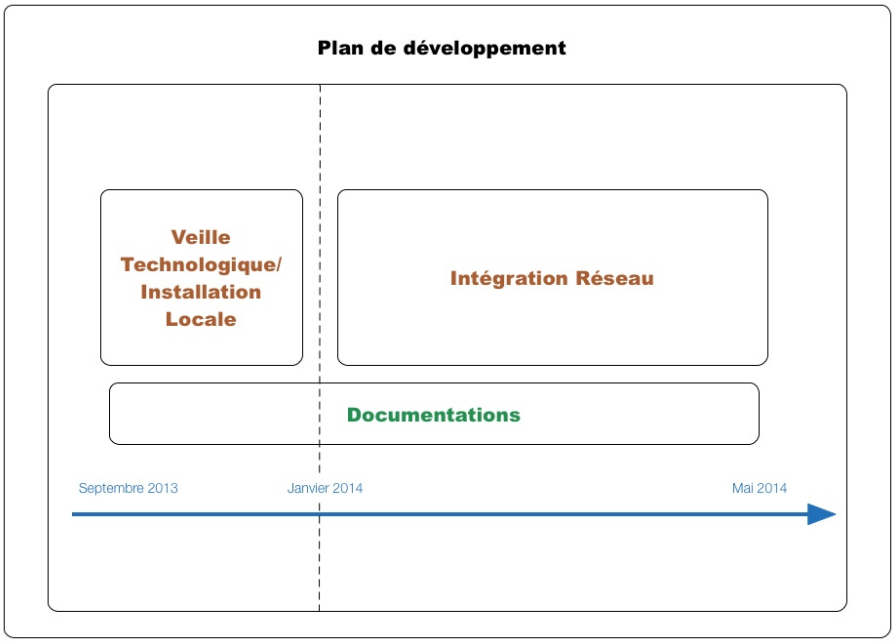

FIGURE 4.2 – Plan de développement du projet

résultat final étant en accord avec l'ensemble des caractéristiques présentes dans le cahier de spécifications : L'ensemble des langages demandés, à savoir C/C++, Java/Maven, C#, a été testé sous forme de projets sous le serveur et dans les délais impartis.

4.2 Communication

Je me permets de faire un paragraphe dédié à la communication qui a été un point très important dans ce projet. En effet, j'ai, contrairement aux années passées, essayé d'être la plus rigoureuse possible en matière de communication.

J'ai tout d'abord mis en place un système de rapport sous forme de mails dans lesquels je décrivais chaque semaine mes avancées. Ayant deux encadrants (Mr T'Kindt et Mr Ragot) ainsi qu'un intervenant (Mr Meichel), c'était le moyen le plus efficace de tenir au courant tout le monde de l'état du projet. J'ai cependant du parfois venir trouver en urgence mes encadrants lorsque je me suis retrouvée face à de "réels problèmes", comme la migration du serveur ubuntu vers windows...

Durant ce projet, j'ai également mis en place des réunions. Compte tenu du nombre d'encadrants, il n'a pas toujours été facile de fixer des dates convenant à tout le monde, et pour cette raison, les réunions n'étaient là que pour traiter les évènements les plus importants. Ainsi, on peut comptabiliser une à deux réunions par étapes de projet (étapes présentées auparavant dans la partie planification du projet). Elles avaient pour but de conclure un travail passé, de vérifier ce qui avait été réalisé (si l'on était sur le bon chemin), de lever les blocages rencontrés et d'orienter le travail vers de nouvelles évolutions.

Notons que le serveur Redmine n'a pas été utile dans mon cas au sens conventionnel du terme. En effet, bien que l'ayant utilisé tout au long de mon projet pour mener mes tests (le serveur développé devait en effet etre raccordé au serveur redmine afin de pouvoir utiliser le serveur Subversion associé), il n'a pas été utilisé afin de partager du "code" avec mes encadrants, puisque je ne travaillais pas sur un projet développement type...

Bilan personnel

Ce projet de fin d'études m'a énormément apporté, que ce soit au niveau des connaissances (théoriques et pratiques), ou que ce soit au niveau personnel. Dans cette partie je vais détailler les connaissances et savoir-faire acquis durant ce projet ainsi que leurs conséquences, avant de terminer sur un bilan plus personnel de cette aventure.

5.1 Connaissances et savoir-faire

Evidemment, et comme son nom le mentionne, le projet de fin d'études "Mise en place d'un serveur d'Intégration Continue et Qualité de Code" m'a énormément apporté de connaissances d'un point de vue "Génie Logiciel" et a complété la formation que j'ai pu avoir durant ces trois années au sein de Polytech Tours.

Tout d'abord j'ai appris à me débrouiller et à approfondir mes connaissances dans différents systèmes d'exploitation : A la base je ne connaissais vraiment que le système d'exploitation Mac OS, sur lequel je développe depuis mon entrée à Polytech pour l'ensemble de mes projets, mais ce projet-ci m'a amené à installer mon serveur sur différents OS afin de tester le pour et le contre : J'ai ainsi appris à utiliser plus en profondeur les systèmes d'exploitation Ubuntu et Windows 7 (choix final pour le serveur...). Cela n'aurait pas été le cas si j'avais uniquement été amenée à devélopper un projet dans un langage type !

De plus ce projet m'a permis de mettre en pratique des savoirs déjà présents, notamment ceux de conduite de projets et de génie logiciels, à travers la mise en place de nombreuses réunions et d'un suivi régulier avec mes encadrants Mr Ragot et Mr T'Kindt. Bien que n'ayant pas eu l'occasion de mettre en pratique ces savoirs durant mon PIL (pour cause de séjours Erasmus en Pologne), je pense que l'on peut quand même dire que ce fut un succès (d'un point de vue conduite de projets) pour cette première mise en pratique.

J'ai aussi pu comprendre et mettre en pratique dans des cas concrets l'application de méthodes dites d'Intégration Continue et de Qualité de Code, grâce aux outils installés sur mon serveur : Jenkins, SonarQube (outils détaillés dans la partie précédente 4.1.1) ainsi que l'utilisation de la plateforme Redmine (avec l'utilisation du gestionnaire de code source Subversion) de l'école.
Je peux d'ailleurs affirmer connaître et maîtriser ces 3 outils, à savoir Jenkins, SonarQube, SVN, ce qui va me permettre d'ajouter d'autres cordes à mon arc et d'avoir un profil plus recherché par les entreprises : En effet, les technologies d'intégration continue et qualité de code sont de nos jours très utilisées par les grandes entreprises et de ce fait les personnes ayant des compétences dans ces domaines possédent un atout non négligeable...

5.2 Se découvrir ...

Ce projet de fin d'études, plus encore qu'un apport sur le plan technique, m'a permis de me découvrir des ressources et capacités que je ne connaissais pas.

Pour ce projet, je me suis vraiment investie, tentant de résoudre des problèmes quitte à m'acharner dessus de nombreux jours (voir même des semaines !), ce dont je n'étais pas capable avant (préférant trouver des

excuses plutôt que de chercher une réelle solution...).

J'ai également du me forcer à m'imposer une certaine rigueur quant à ce projet de fin d'études : Je me souviens de certains projets, où je ne rencontrais les encadrants que quelques heures pour plusieurs mois de travail. Ce ne fut pas le cas ici : Tout au long de mon projet je me suis forcée à dresser des rapports à mes encadrants, chaque semaine, de l'avancée du projet, des retards possibles (ou au contraire des avancées qui me faisait gagner du temps). Cette rigueur m'a appris à rebondir de nombreuses fois, lorsque je me sentais bloquée, à aller poser des questions, et vaincre ma timidité...

J'ai aussi appris à me remettre en question et à accepter le changement, ce qui n'était pas forcément évident lorsque celui-ci impliquait d'effacer des dizaines d'heures de travail (comme le changement de plateforme ubuntu vers windows par exemple, ou j'ai du procéder à la réinstallation complète de mon serveur sur une plateforme que je ne connaissais que peu...).

De plus, ce projet m'a amené à devoir produire énormément de documentations, afin de réussir à justifier l'ensemble de mes choix. Cela n'a pas été facile pour moi à admettre au début, de devoir justifier l'utilisation de tel ou tels outils (Jenkins plutôt que Tinderbox par exemple) mais cela m'a permis de partir sur des bases saines et claires et finalement de gagner du temps lors de l'installation du serveur.

Enfin, et pas des moindres, j'ai la satisfaction d'un travail vraiment abouti (même si je reconnais qu'il reste des choses à faire et que le serveur reste perfectible) et d'avoir su surmonté les nombreuses difficultés qui se sont dressées devant moi durant cette dernière année d'études. Je sais que ce que j'ai fait fonctionne et sera utile aux élèves de l'école Polytech Tours : Je pense ainsi avoir rempli ma mission.

Conclusion

Même si tout n'a pas été toujours rose, je me suis vraiment épanouie dans ce projet. La satisfaction d'un travail abouti (tel qu'il a été défini dans le cahier des charges) est une belle conclusion de ces cinq années passées au sein de Polytech Tours : Ce travail de longue haleine m'aura permis de mettre en pratique l'ensemble des enseignements que j'ai pu recevoir dans cette école durant mes études.

Je reconnais avoir eu la chance de travailler sur un sujet qui me tenait à coeur, à savoir le génie logiciel, et actuel (l'intégration continue est un phénomène extrêmement récent) et qui m'a permis de me former à des technologies recherchées par les entreprises à l'heure d'aujourd'hui.

Je tiens une fois de plus à remercier mes encadrants pour m'avoir permis de travailler sur ce projet de "Mise en place et configuration d'un Serveur d'Intégration Continue et Qualité de Code" et de m'avoir soutenu et aidé à achever ce projet.

École Polytechnique de l'Université de Tours
64, Avenue Jean Portalis
37200 TOURS, FRANCE
Tél. +33 (0)2 47 36 14 14
www.polytech.univ-tours.fr

Département Informatique

Cahier de spécification système & plan de développement			
Projet :	**Mise en place d'un serveur d'Intégration Continue et Qualité du Code**		
Emetteur :	Anne Castier	Coordonnées : anne.castier@etu.univ-tours.fr	
Date d'émission :	16 avril 2014		
Validation			
Nom	Date	Valide (O/N)	Commentaires

Vincent T'Kindt : 12/01/2014 ; N ; **Nicolas Ragot :** 10/01/2014 ; N ;

Historique des modifications		
Version	Date	Description de la modification

00 : 30/10/2013 ; Version initiale
01 : 10/01/2014 ; Version modifiée
02 : 14/01/2014 ; Version modifiée
03 : 21/01/2014 ; Version finale

Table des matières

Cahier de spécification système

1.1 Introduction

Ce cahier de spécification constitue le document de référence pour le projet de fin d'études : *"Installation et Configuration d'un Serveur d'Intégration Continue et de Qualité de Code"*, choisi par l'élève Anne Castier et encadré par Mr Vincent T'Kindt et Nicolas Ragot.
Ce cahier de spécification définit les besoins ayant nécessité la mise en place du projet, et spécifie les fonctionnalités de la solution technique choisie ainsi que les contraintes auxquelles cette solution est soumise. Ce document dévoile également le plan de développement du projet auquel l'élève est tenu de se tenir pour mener à bien son projet de fin d'études.

1.2 Contexte de la réalisation

1.2.1 Contexte

Le but de ce projet est de fournir un outil supplémentaire aux élèves développant d'importants projets informatiques (comme les PILs) dans le cadre de leurs études au sein de l'université François Rabelais.
Le personnel de l'université, ainsi que les chercheurs, doctorants, seront également à même d'utiliser cet outil.

Cet outil sera constitué d'un ensemble d'instruments et mécanismes permettant la mise en place de bonnes pratiques appelées *"Intégration Continue"* et *"Qualité de Code"*.
Nous reviendrons par la suite sur la nature et la mise en place de cet outil.

1.2.2 Objectifs

L'objectif principal de ce projet est d'installer et configurer une machine virtuelle (à porter sur un serveur) au niveau de l'école, regroupant divers outils et logiciels couramment utilisés dans le cadre d'utilisation des procédures d'intégration continue et de qualité de code.

Cependant due à la multitude d'outils et plugins disponibles permettant de personnaliser à l'infini l'environnement de développement nécessaire à l'élaboration de projets spécifiques, il conviendra dans ce projet de faire des choix quant à l'utilisation ou non de certains plugins et logiciels. Ce sera notre premier objectif intermédiaire, synthétisé par notre veille technologique.

Notre second objectif intermédiaire constituera à l'installation de la machine virtuelle et sa prise en main locale.

Enfin, notre dernier objectif sera l'intégration de la machine virtuelle au sein du réseau, et l'ensemble de tests qui s'y rapportent. De plus, afin d'orienter et d'aider l'utilisateur pour une bonne utilisation de ce serveur et des outils disponibles, il conviendra de mettre en place une documentation complète ainsi que des guides.

1.2.3 Hypothèses

L'un des premiers facteurs susceptibles de retarder fortement concerne les contraintes qui sont imposées tant au niveau hardware (choix du serveur, choix des machines allant supporter la machine virtuelle...) qu'au niveau software (choix du système d'exploitation du serveur), données par l'Université (qui possède déjà son prope réseau).

Une réunion prévue le 21/11/2013 avec le service informatique permettra d'en discuter et de poser des bases saines quant à l'exécution de ce projet.

Par exemple, il convient de se poser des questions sur le choix du serveur de gestion de versions. Des serveurs SVN étant déjà mis à disposition à l'école Polytech Tours, il conviendra peut-être de s'appuyer dessus. Si cela n'est pas possible, on mettra en place nos propres serveurs de gestion de versions.

Notons cependant que l'ensemble des outils est open-source, et en règle générale gratuit ce qui devrait constituer un point positif et permettre plus facilement leur mise en place au sein de l'Université.

Une autre hypothèse concerne le fait que la plupart de ces outils sont fortement orientés Java/J2EE. L'un des objectifs de ce projet étant de permettre aux élèves de développer dans de multiples autres langages (C/C++, C#, pour ne citer qu'eux), il convient de trouver une solution permettant de passer outre cet aspect.

1.2.4 Bases méthodologiques

Pour ce projet, il convient de définir clairement ce que l'on appelle " Intégration Continue " et " Qualité de Code ".

> *" L'intégration continue est un ensemble de pratiques utilisées en génie logiciel consistant à vérifier à chaque modification de code source que le résultat des modifications ne produit pas de régression dans l'application développée. "* ***Wikipédia.***
>
> *" L'intégration continue a ainsi pour but de faire de l'intégration une procédure la plus automatisée possible et ceci afin de réduire la durée et l'effort nécessaire à chaque épisode d'intégration. "* ***Institut Agile.***

Intégration Continue

Par intégration on entend : " *Les étapes nécessaires à la production de toutes les données nécessaires à un projet et leur assemblage pour rendre le projet fonctionnel* ". Ainsi si l'on prend l'exemple de deux développeurs ayant travaillé sur deux composants A et B en parallèle, le fait de corriger A et B de manière à ceux que ceux-ci s'interfacent correctement relève de l'intégration.

Ainsi l'intégration continue nécessite la mise en place d'un certain nombre d'outils :

- Logiciels de gestion de version du code source (Subversion, Git, CVS...) : Ces outils permettent de partager le code source entre développeurs.
- Outils d'automatisation de la compilation et génération du projet (Maven, Ant..)
- Serveur d'intégration continue (CruiseControl, Jenkins, Tinderbox...)
- Mise en place de tests unitaires et fonctionnels (JUnit, CUnit, cppunit)

.

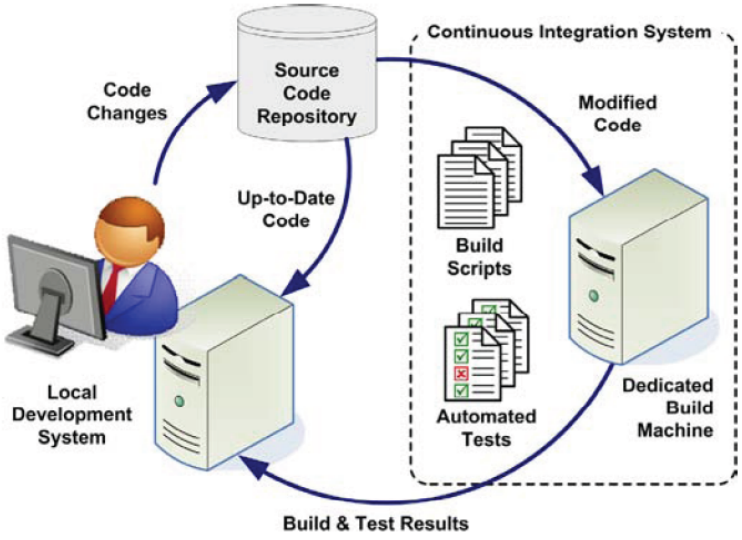

FIGURE 1.1 – Intégration Continue, fonctionnement

Quelques explications générales sur le précédent schéma s'imposent :

Sur ce schéma nous distinguons nettement une partie cliente et une partie serveur. L'utilisateur " commit " son projet sur le serveur de gestion de version du code source, après l'avoir buildé à l'aide d'outils d'automatisation de la compilation. Chaque événement sur ce serveur déclenche le lancement des tâches crées sur les serveurs d'intégration continue.. L'utilisateur peut ainsi voir si il y a eu régression du code après son " commit ".

Nous détaillerons la liste des outils choisis pour l'intégration continue, parmi la multitude proposée, dans le cadre de ce projet et les raisons de ces choix.

Qualité de Code

Quant à la qualité du code, il s'agit - comme son nom l'indique - d'outils pouvant indiquer l'état du projet (couverture proposée des tests, lignes de code en redondance...), c'est à dire la qualité du code.
.

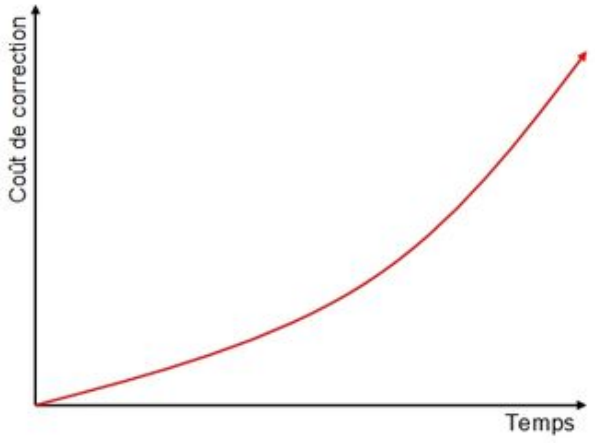

FIGURE 1.2 – Le coût de correction d'une erreur croît exponentiellement avec le temps !

Pour finir, rappelons que le coût de la correction d'une erreur augmente considérablement avec le temps. L'utilisation de tels outils comme l'intégration continue et la surveillance de qualité de code permet de prévenir l'apparition de telles erreurs et est donc plus qu'une excellente pratique à adopter !

1.3 Description générale

1.3.1 Environnement du projet

Il n'existe à ce jour aucun serveur au sein de l'école Polytech Tours ou au sein de l'université François Rabelais proposant un ensemble d'outils au service de l'intégration continue et de la qualité du code.
Seul un serveur de gestion de versions SVN existe à ce jour et est proposé par l'école Polytech Tours pour l'ensemble des élèves ayant besoin de gérer des projets collectifs conséquents.
L' objectif ainsi est de s'interfacer avec le serveur SVN existant et d'apporter et mettre en place le reste des outils nécessaires.

1.3.2 Caractéristiques des utilisateurs

Le projet va servir à différents types d'utilisateurs que l'on peut séparer en deux catégories :

1. **L'équipe du service informatique qui se charge de la maintenance et l'administration du serveur.**

   — Connaissance de l'informatique : Oui
   — Expérience de l'outil : Oui (Le service informatique de Polytech Tours prend par à ce projet par le biais de réunions, et est susceptible de m'aider en cas de grosses difficultés rencontrées).
   — Utilisateurs occasionnels : Oui : Le service n'interviendra que dans le cadre de maintenance et d'administration de ce serveur.
   — Droits d'accès utilisateurs : Administrateurs. Ils doivent être capables de configurer et administrer le serveur à tout niveau.

2. **Les utilisateurs eux-même (étudiants, enseignants-chercheurs), qui utiliseront le serveur par le biais d'une machine virtuelle préconfigurée.**

   — Connaissance de l'informatique : Oui. Les utilisateurs de cet outil l'utilisent dans le cadre de développement de projets informatiques assez complexes (PIL, par exemple) et ont de ce fait des compétences dans le domaine informatique.
   — Expérience de l'outil : Non. Il se peut que les utilisateurs n'aient pas le droit à de formations détaillées sur l'ensemble de ces outils pouvant être complexes. Ainsi la documentation devra être la plus complète possible afin de guider le mieux possible les utilisateurs dans l'utilisation de cet outil.
   — Utilisateurs réguliers : Ces utilisateurs sont supposés utiliser cet outil tout au long du développement de leur projet, qui peut durer plusieurs mois, et s'étaler jusqu'à plusieurs années...
   — Droits d'accès utilisateurs : Utilisateurs. Ils peuvent déposer et récupérer leurs sources en utilisant un outil de gestion de version du code source, configurer leur projet sur le serveur d'intégration continue.

1.3.3 Fonctionnalités et structure générale du système

Ce schéma s'articule sur deux parties indispensables :

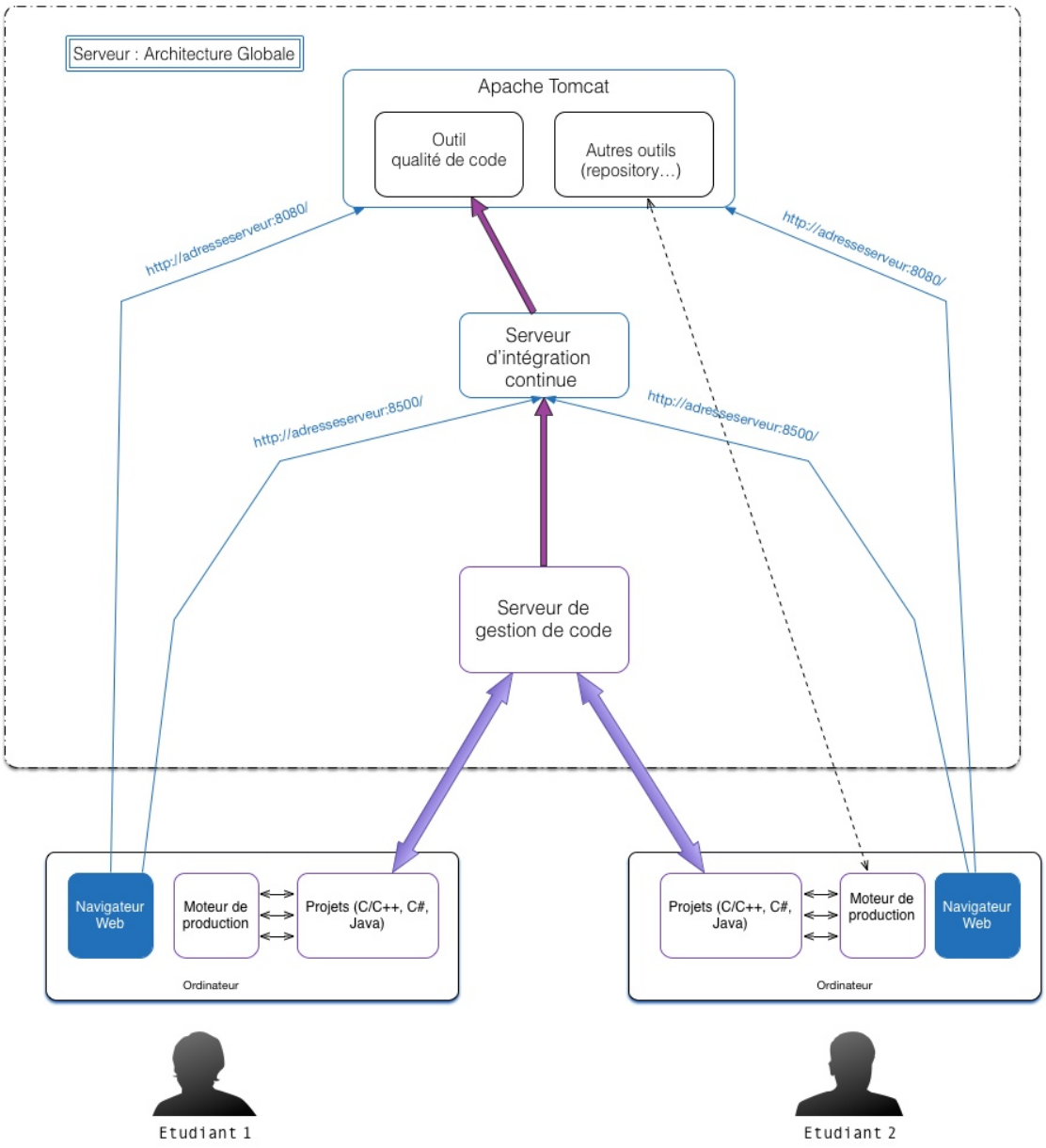

FIGURE 1.3 – Architecture générale du serveur

1. Un environnement client qui comporte au minimum :

   — Un EDI (Environnement de développement intégré).
   — Un moteur de production, outil pour la gestion et l'automatisation de production des projets : L'objectif est de produire un logiciel à partir de ses sources, à l'instar d'un make.
   — Un logiciel de gestion de versions (si possible SVN, pour pouvoir s'interfacer avec ce qui existe déjà au sein de l'établissement).
   — Un navigateur web (**Firefox** de préférence), ainsi qu'une connexion internet.

2. Un environnement serveur constitué de :

   — Un conteneur web libre de servlets. Il servira de conteneur à l'ensemble des outils mis à dispo-

sitions par le serveur.
- Un outil de d'évaluation de qualité de code, logiciel libre permettant de mesurer la qualité du code source en continu (supportant les langages de programmation qui nous intéressent : Java, C/C++, C#.)
- Un serveur d'intégration continue dont le choix sera fait à la suite de la veille technologique.
- Un serveur de gestions de versions, dont le choix sera fixé à l'issu de la première réunion avec le service informatique de Polytech Tours.
- Un ensemble d'outils nécessaires à la mise en place des autres composants du serveur. Cet ensemble sera évalué lors de la veille technologique.

RQ : L'ensemble des éléments choisis caractérisant cette architecture sera détaillé par la suite (cf. parties 1.5 et 1.6)
Il est également possible que le projet soit légèrement modifié pour adopter la forme suivante :

.

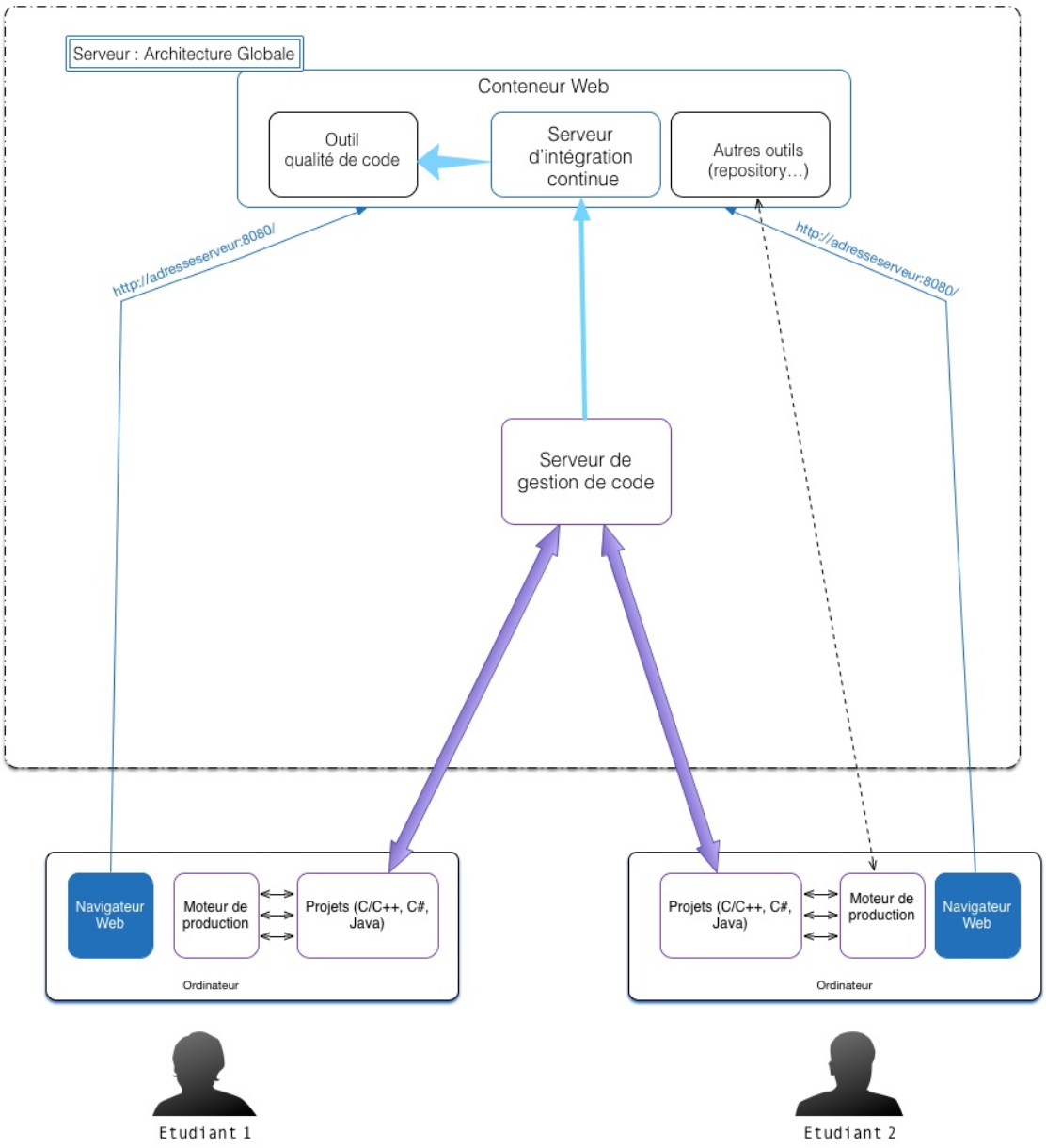

FIGURE 1.4 – Architecture générale du serveur

En effet, il est peut-être possible d'implémenter le serveur d'intégration continue de manière efficace dans le conteneur web. (Ce qui simplifierait la prise en main par l'utilisateur).

1.3.4 Contraintes de développement, d'exploitation et de maintenance

Contraintes de développement

L'ensemble des contraintes a été définie ou abordée durant la réunion du 21/11/2013 avec monsieur Pascal Meichel de l' équipe du service informatique de Polytech'Tours, monsieur Vincent T'Kindt, ainsi que le responsable des projets de fin d'études monsieur Nicolas Ragot.

1. Comme précisé auparavant le système devra être capable de fournir l'ensemble des outils choisis aux projets développés et ceci dans les langages suivants : **C/C++, C#, Java**. C'est en effet les

langages généralement utilisés dans des projets tels les PILs.

2. Le projet devra s'appuyer sur le serveur svn existant plutôt que d'en créer un autre.

3. Il est évident de plus que le serveur devra proposer un ensemble d'outil **simple d'utilisation** et **complet** permettant la mise en place des bonnes pratiques de l'intégration continue et de qualité de code. **Le choix de l'ensemble de ces outils est détaillé dans la veille technologique présentée dans le document** C.

4. L'ensemble des délais de réalisation, ainsi que les étapes nécessaires pour ce projet sont entièrement décrites dans la deuxième partie de ce document : Plan de développement.

Contraintes d'exploitation

Les contraintes d'exploitations portent notamment sur la capacité du serveur. En effet celui-ci doit permettre de stocker les données récupérées par l'ensemble de nos outils d'intégration continue et qualité de code et doit pouvoir supporter plusieurs centaines de projets (anciens projets, projets en cours).
Le serveur doit pouvoir être en service durant l'année entière, 24h/24 à l'ensemble des étudiants de l'école en ayant besoin. *Rappelons que l'école Polytech Tours accueille en son sein plus de mille étudiants et est le siège de quatre laboratoires de recherche importants de l'Université de Tours, dont plusieurs susceptibles d'utiliser l'environnement mis à disposition.*
Notons de plus, en fin de semestre (équivalent souvent à la fin des projets), **la charge portée sur le serveur peut être très conséquente** : l'ensemble des projets présents sur le serveur seront en effet sollicité par l'ensemble des élèves désireux de fournir un projet fini et complet.

Concernant l'affectation des responsabilités utilisateurs : On divisera les droits accordés aux utilisateurs en deux parties : Administrateurs et Utilisateurs.
Les administrateurs auront tous les droits sur l'ensemble du serveur et les outils qui s'y rattachent ainsi que le devoir de créer des comptes pour les futurs utilisateurs (les élèves possédant par ailleurs de base des comptes définis dans l'active directory de l'école ainsi que d'un compte SVN) : Les utilisateurs auront des droits plus spécifiques restreints à l'utilisation de leurs différents comptes.

En effet, il ne faut pas qu'un intrus puisse accéder aux données d'un projet et supprimer ces données (par exemple supprimer les informations liées à un build jenkins, ou pire : supprimer/modifier des sources du projet).

Il faudra également s'attacher à permettre la sauvegarde des données présentes sur le serveur de manière régulière. L'ensemble de ces contraintes devra également être appuyé d'une forte documentation, afin de pallier tous les cas possibles d'utilisations et de proposer des solutions à tout problème pouvant être engendré.

Maintenance et évolution du système

Le service informatique de Polytech Tours sera "chargé" de la maintenance et évolution du système. Par exemple il se peut qu'un élève pour l'un de ses projets ait besoin d' évolutions des outils mis à disposition par le serveur. Ainsi l'équipe du service informatique sera à même d'effectuer ces évolutions (ajouts de plugins, etc...). Cependant, ces besoins d'évolutions doivent rester rares, d'où l'importance de la veille technologique (voir document C).

1.4 Conditions de fonctionnement

Dans ce paragraphe, je décrirais les dispositions qu'il est nécessaire de prendre en compte pour les différentes conditions de fonctionnement du système. **Cette section est à confirmer et à développer à l'aide des appréciations du service informatique.**

1.4.1 Performances

Comme dit précédemment, il est essentiel que le serveur supporte de grosses montées de charge : En fin d'année, ou à chaque période de rendue de projets, des groupes d'élèves (de 2 à 10 élèves suivants les projets) se connecteront simultanément au serveur pour bénéficier de l'ensemble des outils présentés précédemment.

Cependant, il est difficile pour moi de définir concrètement en termes mesurables des spécifications temps réels pour le serveur à mettre en place. Il faudrait pour cela avoir déjà une préexpérience dans la mise en place de serveurs.
On peut cependant se fixer des cas limites
.

— **Du point de vue de l'utilisateur :**

1. Temps de réponse acceptable : Plus le délai est court, plus l'interface sera agréable d'utilisation. Fixons-nous cependant une première base de 1s de temps de latence moyenne.
2. Fréquence d'utilisation : Par "groupe" d'étudiants, une fréquence moyenne peut être estimée à maximum 8h/semaine.
3. Temps d'indisponibilité acceptable : De 2 à 8h, si possible durant la nuit, en début d'année scolaire.

— **Du point de vue de l'environnement :**

1. Fréquence maximale d'E/S : On se fixera une fréquence maximale d'entrée/sortie de 1000/minutes, l'école de Polytech Tours accueillant en son sein plus de 300 (nombre approximatif) élèves travaillant simultanément sur des projets susceptibles d'utiliser le serveur d'intégration continue et de qualité de code.

1.4.2 Capacités

La capacité est l'un des points cruciaux de ce serveur. Cependant, l'un des faits évoqués durant la première réunion avec l'équipe du service informatique, est la possibilité de faire évoluer le serveur assez facilement grâce à l'utilisation d'une machine virtuelle.
.
Cependant fixons-nous une base de départ :

— **Capacité max de stockage** : Le serveur doit être capable de stocker sur plusieurs années des projets volumineux (types PILs) pour des centaines d'étudiants. Ces projets peuvent peser plusieurs GO de données... Ainsi une base de 5 To est acceptable.
— **Taille max des données traitées** : Les outils tels Jenkins, SonarQUbe... sont amenés à traiter énormément de données générées par des projets extrêmement volumineux. On fixera une limite (théorique) à 10 Go de données...
— **Latence** : Une latence de 1s sera tolérée au début.

1.4.3 Modes de fonctionnement

Pour ce projet, et à ce niveau, on ne considérera pas de fonctionnement dégradé. Un seul mode de fonctionnement, dit normal, sera considéré.

1.4.4 Sécurité

On avait déjà vu que l'on avait deux types d'utilisateurs :
— Administrateurs
— Utilisateurs

Nous allons autant que possible nous baser sur l'active directory déjà mis en place pour gérer l'ensemble des utilisateurs et administrateurs. Cette partie interviendra à la fin de la mise en place du serveur d'intégration continue *(cf plan de développement)*. Pour plus d'informations, il faut se référer à la partie 3.2, caractéristiques des utilisateurs.

1.4.5 Intégrité

Dans cette partie on précise les protections contre la déconnexion imprévue, les pertes d'information... et quelles sont les procédures à suivre pour restaurer les données du système.

Pour ce projet de fin d'études, il faudra envisager de considérer deux types de backup :

1. Backup des données
2. Backup des services

Il faut penser à faire confirmer cette section par Monsieur Pascal Meichel, membre de l'équipe du service informatique de Polytech Tours.

1.5 Architecture générale du système

Nous partons sur un modèle d'architecture basique client/serveur.

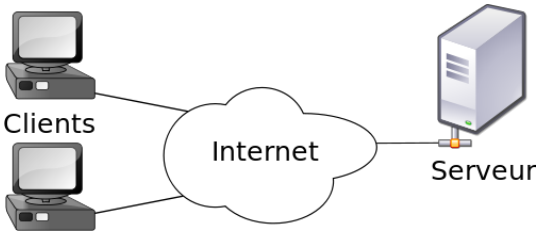

FIGURE 1.5 – Architecture type client/serveur

Toutes les spécificités liées à l'environnement du serveur et de la machine cliente seront expliquées par la suite.

1.6 Description des fonctionnalités et logiciels utilisés

Dans cette partie nous détaillerons l'ensemble des logiciels retenus (suite à la veille technologique présente en annexe dans le document C), leur fonctionnement, leurs façons d'interagir entre elles. Nous détaillerons en détail les solutions techniques choisies qui permettront de répondre aux demandes du cahier des charges (par exemple, les plugins installés et leur importance).

— Jenkins
— Apache Tomcat 7
— SonarQube
— Eventuellement, Nexus Sonatype.

Celles-ci se répartiront de la façon suivante (pour faire le parallélisme avec la première partie de ce cahier de spécifications) :

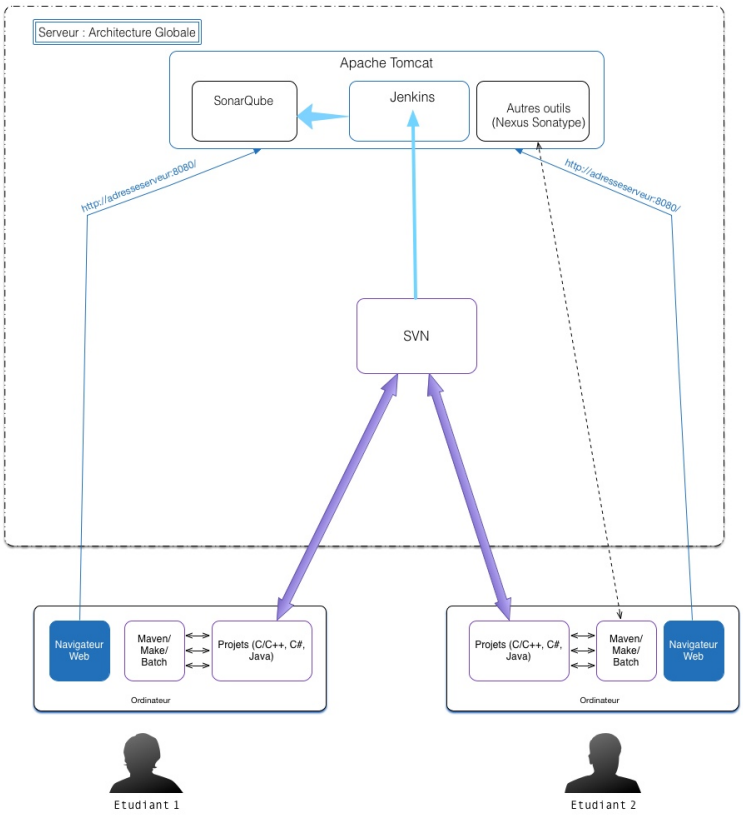

FIGURE 1.6 – Architecture générale du serveur

1.6.1 Jenkins

Description de Jenkins

FIGURE 1.7 – Jenkins Logo

Le choix de Jenkins s'est fait en connaissance de cause suite à la veille technologique présente en annexe dans le document C.
Cette partie va permettre la description de Jenkins qui devra être proposé à l'utilisation depuis notre serveur ainsi que la décision finale choisie quant à sa configuration et les plugins choisis.

Description

Jenkins est un serveur d'intégration continue très populaire (surtout pour les projets développés en java à l'aide du moteur de construction Maven, bien qu'il fasse l'affaire pour de très nombreux autres

langages...). Il s'appelait autrefois Hudson, mais a été renommé suite à un différent entre Oracle, société détentrice du nom, et la communauté open-source.

Jenkins possède deux rôles principaux :

1. Lancer des compilations et tests de projets logiciels, et ce de manière continue
2. Gérer des exécutions d'autres tâches externes (par exemple, pour nous, SonarQube !)

Il est extrêmement utilisé dans le domaine des entreprises (ce qui permet ainsi aux élèves de se former à des technologies prisées par le monde extérieur à l'école).

Plugins choisis

Jenkins ne propose pas moins de **600** plugins pour étendre ses capacités et les adapter aux besoins de chacun. Comme pour SonarQube, il convient de choisir ceux qui correspondent le plus aux besoins du projet "Mise en place d'un serveur d'intégration continue".
Cependant, une liste des 600 plugins pour lesquels on devrait poser les pour et contres serait peut être de trop (sauf, si on souhaite atteindre les 100 pages du document veille technologique). Nous nous sommes contentés d'une rapide étude et d'indiquer quels plugins étaient réellement nécessaires.

— xunit plugin
— msbuild plugin
— nant plugin
— cmake plugin
— maven project plugin
— subversion plugin
— sonarqube plugin
— active directory plugin

1.6.2 Apache Tomcat

FIGURE 1.8 – Logo d'Apache Tomcat

Bien qu'il n'ait pas été question d'Apache Tomcat dans la veille technologique, il est recommandé d'installer Apache Tomcat pour le serveur à mettre en place.

En effet celui-ci va servir de conteneur web à Nexus et à SonarQube. Bien que ces deux derniers peuvent s'exécuter en tant que service, il est préférable que le serveur écoute l'ensemble des requêtes sur le même port, pour des soucis de simplification d'usage pour l'utilisateur. *Il est également possible de déployer Jenkins à l'aide de Tomcat, ce que nous ferons si le temps nous le permet. En effet, je n'ai pas eu le temps durant la phase de prise en main de tester Jenkins via Tomcat...*
De plus il est léger, gratuit, rapide, multiplateforme.

L'ensemble des adresses à fournir au navigateur par l'utilisateur sera documenté dans la documentation à l'usage de l'utilisateur.

1.6.3 Moteurs de production : Maven & cie

Nous avons choisi pour ce projet d'utiliser en priorité le moteur de production Maven pour plusieurs raisons, détaillées dans le document C.

FIGURE 1.9 – Logo de Maven

Cependant, (**et pour l'ensemble des raisons détaillées dans le document veille technologie**), l'utilisateur devra, dans le cadre de projets développés en C, C++, C# :

— soit configurer davantage des fichiers de configuration pour l'utilisation de maven
— soit utiliser un moteur de production plus natif ou propre au langage utilisé (batch, NAnt, make...)

Pour plus de précisions, référez vous au document veille technologie (document C.)

1.6.4 Nexus

FIGURE 1.10 – Logo de Nexus

Pour les mêmes raisons qu'évoquées dans le document veille technologie (document C), il est possible et souhaité de choisir Nexus comme repository manager. Celui-ci va nous permettre de rendre la gestion de dépôts Maven plus facile.

1.6.5 SonarQube 4.0

Présentation de SonarQube 4.0

Le choix de SonarQube s'est fait en connaissance de cause suite à la veille technologique présente en annexe dans le document C.

Cette partie va permettre la description de SonarQube 4.0 qui devra être proposé à l'utilisation depuis notre serveur ainsi que la décision finale choisie quant à sa configuration et les plugins choisis.

SonarQube est un outil open source dont le but principal est de fournir une analyse complète de la qualité d'une application en fournissant de nombreuses statiques (que l'on appelle aussi métriques) sur des projets informatiques. On peut connaître ainsi tout au long du développement du projet, son évolution, ainsi que la qualité du code engendrée. SonarQube, à lui tout seul, constituera notre outil qui se chargera de la partie "Qualité de Code", c'est dire si cet outil est complet.

Cependant, pour une utilisation de SonarQube qui répondra à toutes nos attentes il conviendra de choisir la bonne configuration, et l'installation des bons plugins dans une juste mesure. En effet, on pourrait croire qu'installer l'ensemble des plugins (plus d'une quarantaine) constitue une solution alternative : cela permettrait d'éviter à l'utilisateur de rajouter par la suite ses propres plugins pour pallier des manques dus

à certaines spécificités de son projet. Mais cela surchargerait inutilement notre serveur, car tous ne sont pas nécessaires. Nous présenterons par la suite la liste des plugins de base retenus.
Note : le site Nemo Sonar Codehaus est mis à disposition pour ceux souhaitant manipuler et observer le comportement de SonarQube à l'aide de gros projets open source déjà établis.

Description de SonarQube 4.0

Pour commencer, il faut bien comprendre le fonctionnement de SonarQube 4.0 et son articulation avec les autres logiciels :

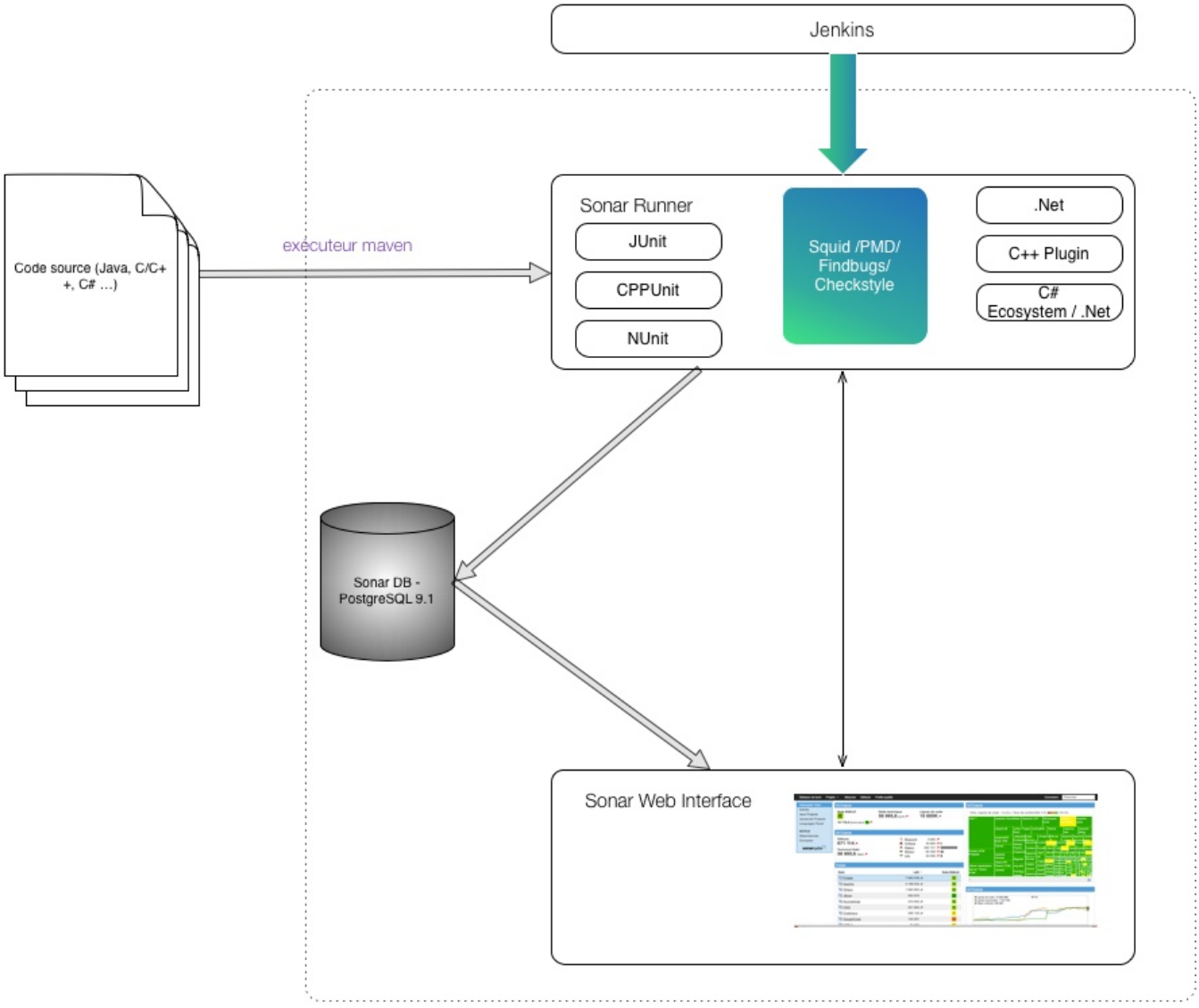

FIGURE 1.11 – Architecture SonarQube

.

SonarQube est constitué de plusieurs couches :

— Notre exécuteur maven2, qui permet à Sonar de récupérer les informations nécessaires pour ses analyses.
— Une base de données où sont stockées toutes les informations des projets observés par Sonar, construite sous PostgreSQL 9.1.
— Une interface web appelée aussi Sonar Web Interface, accessible depuis un navigateur et permettant à l'utilisateur d'accéder à toutes les métriques souhaitées.
— Sonar Runner qui effectue toutes ses analyses à l'aide notamment de "rule engine" : PMD, CheckStyle, FindBugs, Squid.

Cependant, notons que Jenkins va être utile : Après la fin de l'un de ses builds (par exemple), il va permettre de lancer Sonar Runner afin que ce dernier puisse éxécuter ses analyses. Nous verrons cette partie plus tard dans la partie interactions logiciels/logiciels.

Plugins choisis

Il existe actuellement une quarantaine de plugins pour Sonar qui permettent de complèter cet outil et de le configurer pour tout type de projets. Voici la liste des plugins représentant l'ensemble des plugins choisis et présentée sous forme de tableau.

Voici la liste des plugins Sonar, leur utilité, et la décision quant à leur choix final :
— c++ plugin
— c plugin
— c# plugin
— java plugin
— abacus plugin
— build stability plugin
— scm stats plugin
— useless code tracker plugin
— sig maintainability model
— toxicity chart plugin
— pdf report plugin
— ldap plugin
— redmine plugin

Notons cependant qu'une étude bien plus détaillée est présente en annexe dans le document C, mais que les décisions finales ont été prises lors de la soutenance de mi-parcours de PFE (janvier 2014).

1.7 Description des interfaces externes du logiciel

1.7.1 Interfaces matériel/logiciel

Pour ce projet de fin d'études, l'école de Polytech Tours met à disposition de l'étudiant, un serveur sur lequel s'effectuera l'ensemble des développements et tests au cours du projet. Durant la réunion du 21/11/2013, il a été dit que cette machine, due à l'utilisation d'une machine virtuelle, serait facilement reconfigurable en cas de problème (réattribution de mémoire vive, etc.), ou de besoins d'évolutions.

1.7.2 Interfaces homme/machine

Les utilisateurs auront accès à un environnement préconfiguré, une machine virtuelle, pour l'utilisation du serveur constitué de l'ensemble des outils vus en partie 1.3.3.
Voici une interface simplifiée de ce à quoi l'utilisateur a accès :

.

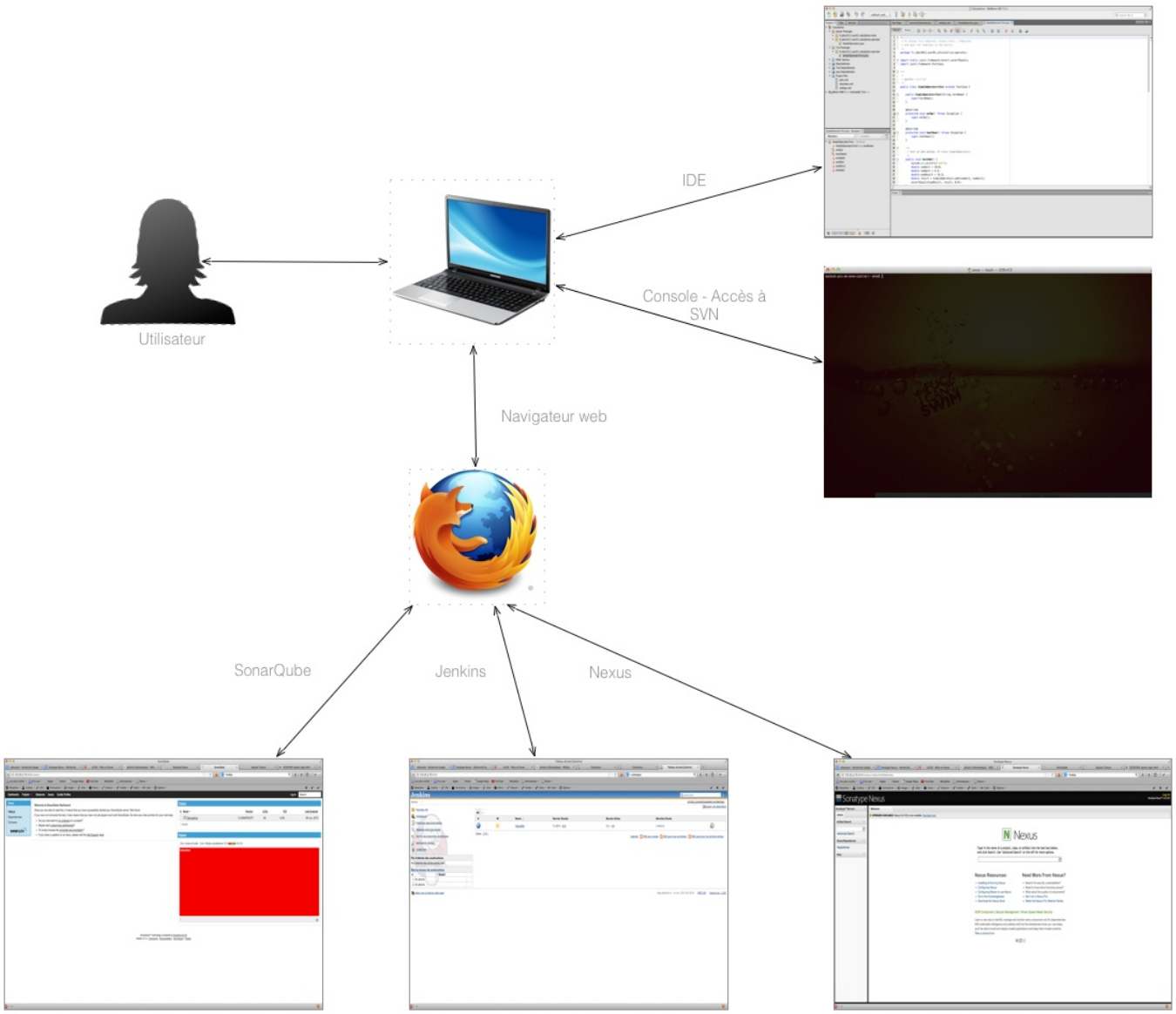

FIGURE 1.12 – Interface Utilisateur

On y distingue bien nos 3 outils principaux délivrés par le serveur (dans l'ordre de gauche à droite) : Sonar, Jenkins, Nexus. Ainsi l'ergonomie du système est de qualité et intuitive puisqu'elle consiste principa-

lement en applications WEB développées par des communautés très actives et très utilisée dans le domaine des entreprises.
Ce ne sont pas des applications lourdes à prendre en main, et elles bénéficient de nombreux tutoriels sur le web. Ces trois outils et leurs fonctionnements seront détaillés en 1.6.
Voici un exemple typique de fonctionnement :

L'utilisateur crée son projet dans le langage java par exemple, en l'architecturant pour qu'il réponde au besoin du serveur subversion. Dans le même temps,l'utilisateur configure un job sur Jenkins de manière à ce que celui-ci puisse lancer les différents builds (maven, MSBuild..). Il le compile ensuite à l'aide de maven (ou batch, selon le langage utilisé), avant de le commiter sur le serveur svn. Il n'a alors plus qu'à aller observer le résultat offert par les 3 outils vu précédemment.

1.7.3 Interface logiciel/logiciel dans le cadre de projets maven

Il faut bien comprendre que cette partie est assez complexe. En effet le but de ce projet est de faire interagir entre eux de nombreux logiciels, et détailler ces interactions est compliqué : De par leur nombre, et de par leur spécificité à chacune. Ainsi, voici les liens généraux qui unissent nos logiciels proposés par le serveur.

.

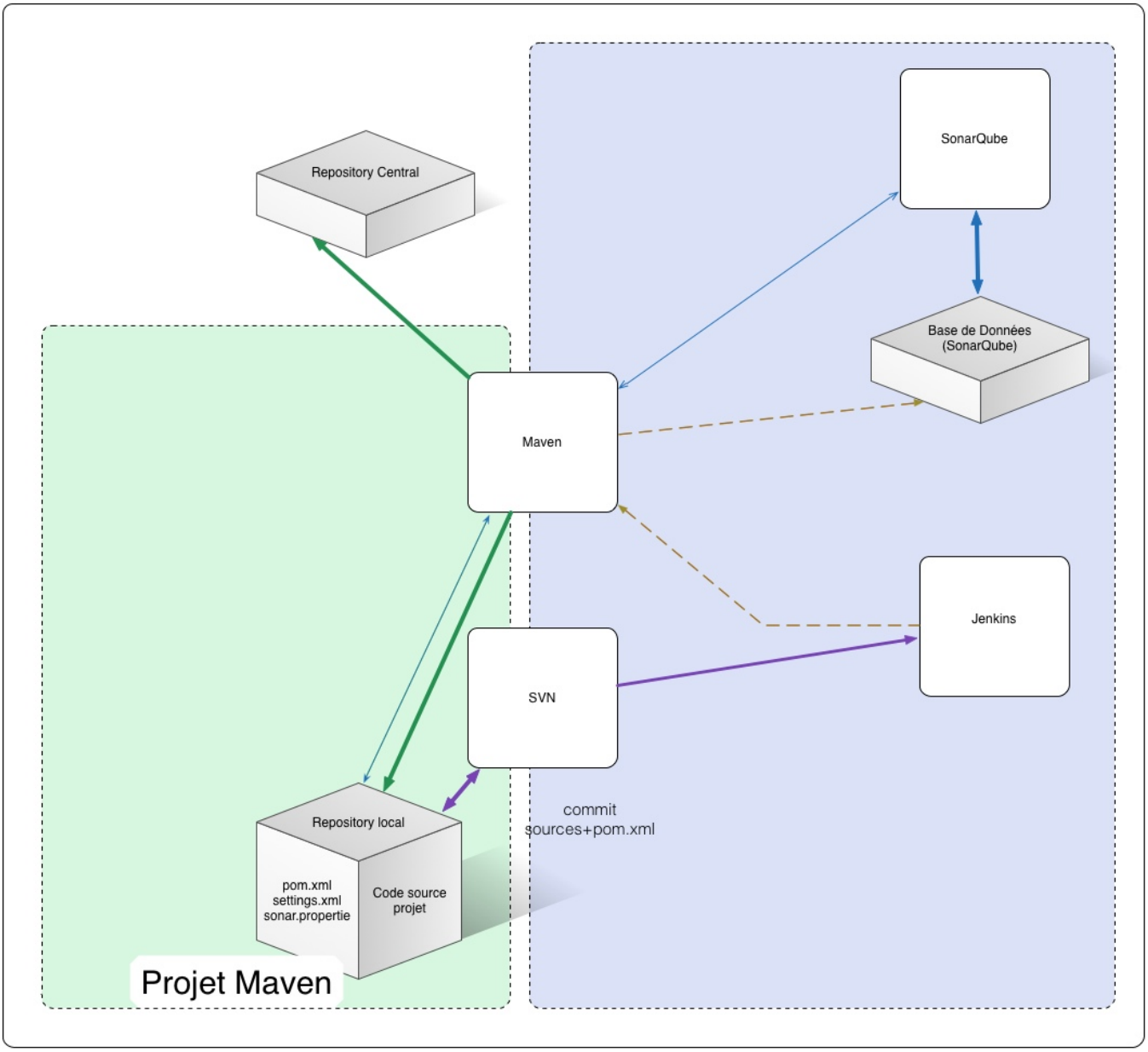

FIGURE 1.13 – Interface Logiciel/Logiciel

Maven

Maven est un framework de scripting de haut niveau orienté Java. De nombreuses fonctionnalités sont disponibles :

- Génération du build/artifact : avec Maven, il est possible de générer le fichier résultant d'un projet Java (.jar, .war ou .ear)
- Gestion aisée des librairies d'un projet Java
- Intégration avec de nombreux serveurs d'intégration continue.

Il se configure à l'aide de deux fichiers : settings.xml (niveau client, permet de gérer l'accès aux adresses où Maven télécharge les librairies (par défault sur Maven Central) et pom.xml(niveau projet, redéfinit le fichier settings pour une configuration plus spécifique du projet). Ces interactions ont été symbolisées par des flèches vertes sur le schéma.

SVN/Jenkins

Jenkins, grâce à ses extensions et plugins, permet de gérer de manière très facile un serveur de gestion de code comme subversion. Lorsque l'utilisateur configure le job concernant son projet sous Jenkins, il configure plusieurs points clés aussi appelés "étapes de build" :

— L' utilisation du dépôt du serveur Subversion voulu(l'url du serveur et des identifiants -si nécéssaires- sont simplement demandés), à savoir là ou se trouve le code source/
— Le lancement automatique (après chaque commit, toutes les heures, chaque nuit...) de build du projet
— Le moteur de production (MSBuild, NAnt, Maven...)

Ainsi, on vient d'expliquer à Jenkins, ou quand et comment éxécuter le code source. Ainsi en lançant un build Jenkins va éxécuter à partir des sources fournies sa tâche de build et fournir les informations demandées (régressions des tests... etc).

SonarQube/Jenkins

De manière générale, SonarQube exécute automatiquement un **ensemble de plugins Maven (Checkstyle, PMD, FindBugs)** liés à la qualité de code à partir du **projet Maven** et dont la configuration est décrite dans le fichier sonar.properties du projet.
SonarQube stocke d'abord les résultats dans une base de données (comme PostGreSQL par exemple), puis analyse ensuite les résultats que l'on peut consulter sur l'interface web. (Cela correspond au **flèches bleues** du schéma). Cette manière de fonctionner est par ailleurs décrite dans la partie SonarQube.

Cependant nous nous intéressons ici à l'intégration de Jenkins avec Sonar. Le plugin Sonar de Jenkins (via l'interface de Jenkins) nous permet de configurer les instances Sonar pour tous nos projets, et les activer par la suite selon la configuration donnée (*lancer une analyse après un build par exemple*). Cependant, Jenkins doit avoir un accès à la **base de données de Sonar**, puisqu il introduit des mesures de qualité de code (Plugins Maven comme CheckStyle, PMD, FindBugs) directement dans celle-ci, *sans passer par l'interface web de Sonar* (cf schéma, **flèches dorées**).

1.8 Livrables

1.8.1 VM côté serveur

L'objectif premier de ce projet de fin d'études est d'aboutir à une machine virtuelle préconfigurée, contenant l'ensemble des outils présentés dans la partie 1.6 et répondant à toutes les exigences données par ce cahier de spécifications. Cette machine virtuelle sera ainsi portée sur le serveur donné par l'école et sera soumis à un ensemble de tests obligatoires.

1.8.2 VM côté client

Une VM client type (que l'on a certes peu évoqué dans ce cahier de spécifications, puisqu'elle ne constitue pas l'objectif final de ce projet de fin d'études) sera également installée. Son objectif sera de permettre à tout un chacun de prendre rapidement en main les outils proposés par le serveur en étant préconfiguré, et en contenant la majeure partie des outils de développement utilisés au sein de l'école (Visual Studio, Codeblocks, Eclipse). Elle sera préconfigurée pour permettre l'utilisation des logiciels du serveur (paramètrage de fichiers maven par exemple).

1.8.3 Documentation

Ces deux livrables seront accompagnés par une documentation la plus complète et la plus compréhensible possible :

- Quick Start Guide (permettant un démarrage rapide). Ce document constituera un tutorial permettant de construire une application basique, créer un build jenkins associé et lecture des informations (une métrique vue en cours de génie logiciel par exemple) dans l'interface de sonarqube. La durée maximale de ce tutorial ne doit pas excéder 2h (Durée d'un TP de génie logiciel).
- Manuel Utilisateurs. Document plus complet, séparé en plusieurs parties (selon le langage utilisé : C/C++, Java, C# ...), et détaillant les possibilités offertes par le serveur.
- Manuel Administrateurs.

.

Il est évident que ce projet sera également accompagné des documents suivants :

- Rapport de PFE
- Cahier de spécifications
- Affiche

Plan de développement

2.1 Découpage du projet en tâches

L'ensemble du projet a été découpé en tâches afin de faciliter son développement et le planifier de façon optimale.

2.1.1 Prise en main et découverte des outils d'intégration continue et qualité de code

Description de la tâche

Cette première tâche a permis à l'élève Anne Castier de prendre en main un ensemble d'outils d'intégration continue et de qualité de code, ce qui lui a permis d'obtenir une première approche du projet à réaliser. Cette tâche s'est soldée par un léger échec sur le test des installations serveur avec un projet c, qui a amené une longue phase de réflexions sur la suite du projet.

Cycle de vie

Cette première tâche s'est ainsi découpée en une liste de sous-tâches :

1. Installation de l'environnement serveur et de l'environnement client (machine virtuelle, java 1.7)
2. Installation des outils dits indispensables à l'intégration continue : maven, ant, svn, git, jenkins, tomcat, sonarqube, et nexus.
3. Tests des installations avec un projet java
4. Tests des installations serveur avec un projet c

Estimation de charge

La tâche a été estimée à 7 jours/homme.

Contraintes temporelles

Cette tâche a été effectuée durant le mois d'octobre du 03 octobre au 31 octobre.

2.1.2 Rédaction Cahier de Spécifications

Description de la tâche

Cette longue tâche a permis de réaliser le cahier de spécification requis, présentant l'ensemble des solutions choisies ainsi que leur plan de développement. Cette tâche a également permis la réalisation de l'affiche nécessaire pour la soutenance de mi parcours.

Livrables

— Cahier de Spécification (CDS)
— Affiche (Poster du PFE)

Estimation de charge

Cette tâche a été évaluée à 10 jours/homme , en raison de la lourde documentation à fournir.

Contraintes temporelles

Le cahier de spécifications doit être rendu avant le 06/01/14.* Le poster du PFE doit être rendu courant janvier.

2.1.3 Veille Technologique

Description de la tâche

Cette tâche a été explicitement demandée par l'ensemble des encadrants de ce projet. Elle permet de justifier l'ensemble des solutions techniques choisies (par exemple, pourquoi l'utilisation de Jenkins ?).

Estimation de charge

Cette tâche a été évaluée à 2 jours/homme.

2.1.4 Installation environnement client/serveur

Description de la tâche

Cette tâche consistera à la configuration de la machine virtuelle à fournir, l'installation et la configuration de l'ensemble des outils choisis et présentés dans ce cahier de spécification. Elle contiendra également la gestion de l'authentification. Toutes ces phases seront amenées à être testées pour l'ensemble des types de projets.

Cycle de vie

— Prise en main du serveur fourni par l'école, préconfiguration de l'environnement
— Installation de l'ensemble des outils choisis sur le serveur
— Première phase de tests
— Gestion de l'authentification
— Seconde phase de tests

Livrables

Cette tâche permettra le livrable d'une machine virtuelle (VM) fonctionnelle supportant l'ensemble des outils proposés, une gestion de l'authentification.

Estimation de charge

Cette tâche a été estimée à 20 jours/homme.

Contraintes temporelles

Le livrable requis doit être rendu avant début mai.

2.1.5 Documentation

Description de la tâche

Cette tâche, qui s'exécutera parallèlement à l'installation environnement client/serveur, permettra d'obtenir une documentation complète permettant à chacun (utilisateurs et administrateurs) d'utiliser correctement le serveur et les outils proposés.

Livrables

Cette tâche permettra d'obtenir un ensemble de documentation permettant à chacun de prendre en main les outils du serveur.

Estimation de charge

Cette tâche a été estimée à 10 jours/homme.

Contraintes temporelles

Le livrable requis doit être rendu avant début mai.

2.2 Planning

Le planning synthétise l'ordonnancement de chacune des tâches pour le projet "Mise en place d'un serveur d'intégration continue et de qualité de code". *Note : Il est prévu une certaine marge à la fin du projet (correspondant au mois d'avril) pour fixer les derniers problèmes et préparer la soutenance. Note : Pour une meilleure lisibilité, ces diagrammes sont également fournis en pièces jointes !.*

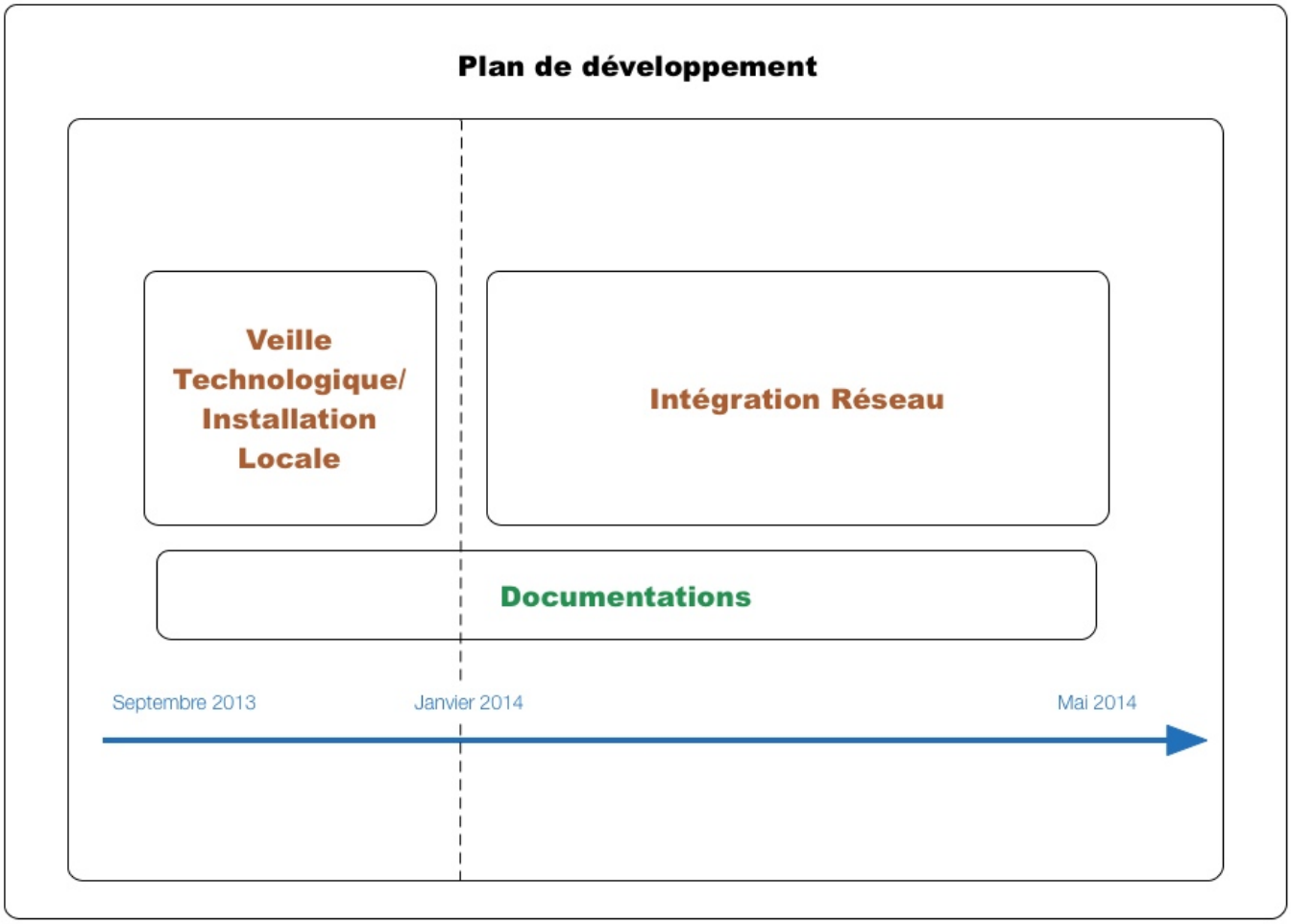

FIGURE 2.14 – Général

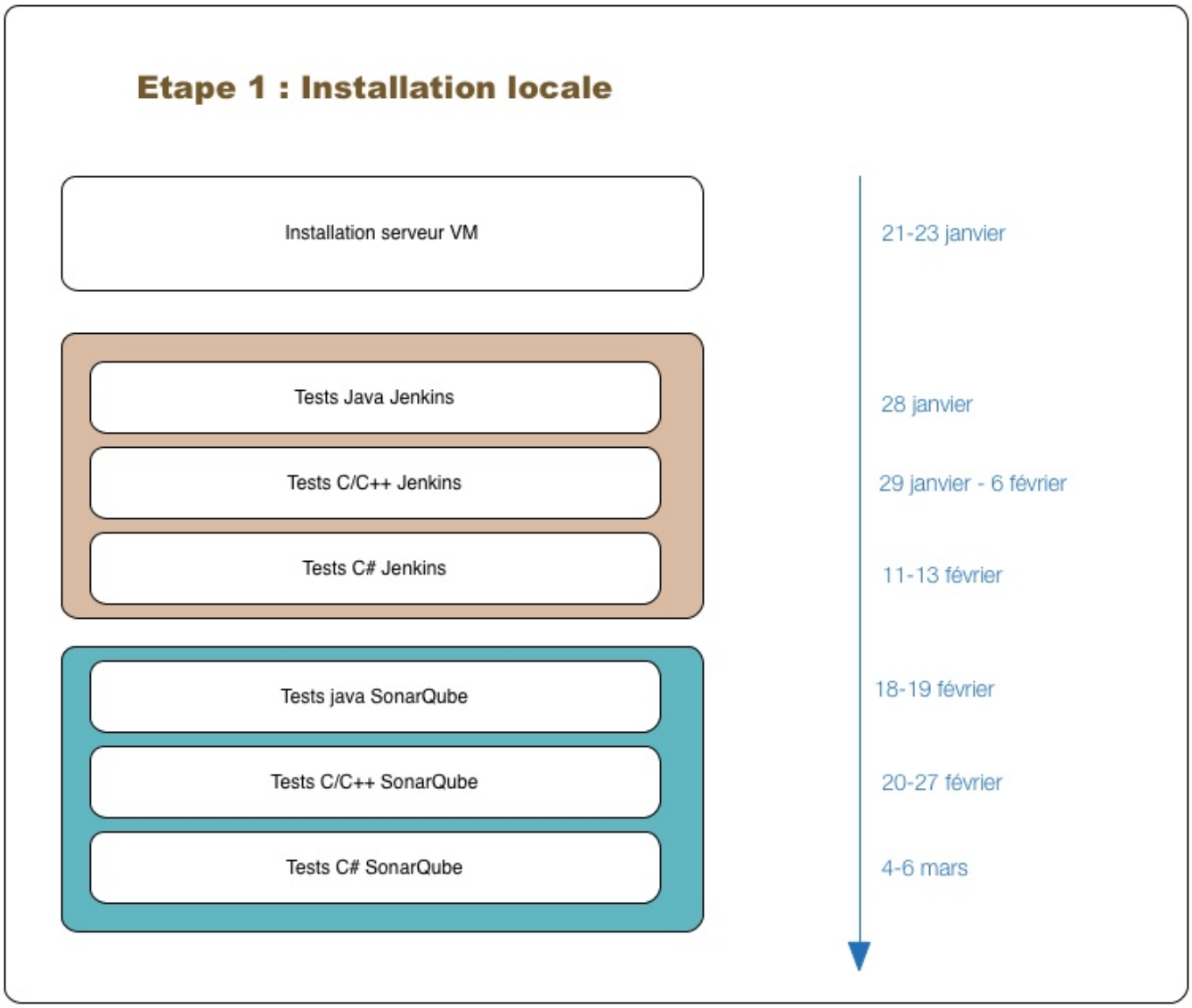

FIGURE 2.15 – Etape 1

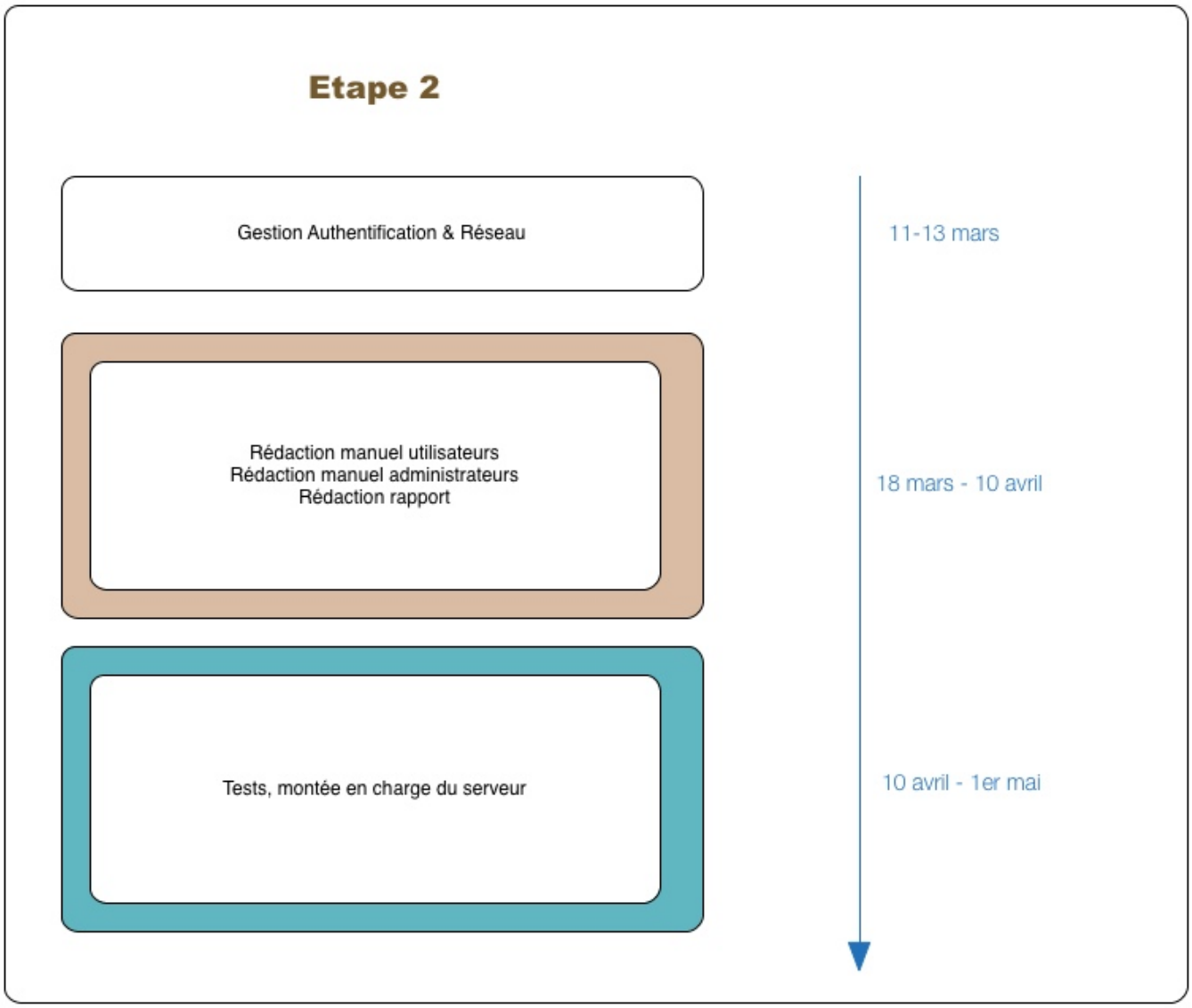

FIGURE 2.16 – Etape 2

Glossaire

Dans cette partie on doit trouver, classés par ordre alphabétique, les définitions des termes courants utilisés, des termes techniques, abréviations, sigles et symboles employés dans l'ensemble du document.

Références

Cette dernière partie recense les références techniques sur le projet sur :
— les documents relatifs à l'existant et à l'environnement ;
— les documents sur les méthodes et algorithmes cités ;
— les documents bibliographiques (internes et externes) ;
— les sources d'obtention des documents.

École Polytechnique de l'Université de Tours
64, Avenue Jean Portalis
37200 TOURS, FRANCE
Tél. +33 (0)2 47 36 14 14
www.polytech.univ-tours.fr

Département Informatique
5e année
2013 - 2014

Veille Technologique

Mise en place d'un serveur d'intégration continue et qualité de code

Encadrants
Vincent T'Kindt
vincent.tkindt@univ-tours.fr
Nicolas Ragot
nicolas.ragot@univ-tours.fr

Étudiants
Anne CASTIER
anne.castier@etu.univ-tours.fr

DI5 2013 - 2014

Université François-Rabelais, Tours

Version du 21 janvier 2014

Table des matières

Introduction

Ce document est à l'origine des décisions prises concernant les solutions techniques choisies dans le cadre du PFE *"Mise en place d'un serveur d'intégration continue et qualité de code"*. Il retrace l'ensemble de la veille technologique effectuée par l'élève Anne Castier et présente ainsi la majorité des outils utilisés dans le cadre des bonnes pratiques *"Intégration Continue"* et *"Qualité de Code"*, leur utilité, leurs avantages et inconvénients et finalement une conclusion sur leur mise en place ou non au sein du serveur d'intégration continue et qualité de code.

Rappel et définitions

La volonté de pratiquer les méthodes d'Intégration Continue (domaine très vaste et pour lequel il est difficile de définir concrètement ses limites) dépend de l'utilisation d'outils spécifiques dont le nombre se compte en milliers.
Il convient ainsi de faire un choix quant aux technologies à utiliser. C'est le but de cette veille technologique, à savoir le choix des bons outils dans un nombre correct, afin de remplir les besoins du cahier des charges.
Pour une meilleure description, et un rappel des bases méthodologiques (nécessaires à la compréhension de cette veille technologique), vous pouvez vous référer à la partie **1.2.4 Bases méthodologiques** du cahier de spécification.

Intégration Continue

3.1 Serveurs d'Intégration Continue

Un serveur d'intégration continue est comme son nom l'indique, un serveur permettant à un utilisateur d'accéder à des outils d'intégration continue, notamment la gestion de régression du code. Il existe de nombreux serveurs d'intégration continue, dont voici une liste non exhaustive :

1. Cruise Control, Cruise Control.NET, Cruise Control.rb
2. Go
3. CI Factory
4. Drumbeat CI
5. Tinderbox & Tinderbox2, Tinderbox3
6. BuildBot
7. Anthill Professional, Anthill
8. Bamboo
9. Luntbuild Professional, LuntBuild
10. Gump
11. Continuum
12. Sin
13. OpenMake Meister, OpenMake Mojo
14. Parabuild
15. Pulse
16. TeamCity
17. Hudson/Jenkins
18. FinalBuilder Server
19. Zed
20. easyCIS
21. RedJack
22. Electric Commander

Pour des raisons de simplification, la veille technologique ne porte que sur une sélection d'entre eux (les plus connus, les gratuits, ceux pouvant supporter au moins java, c/c++ et c#...). Cette veille technologique s'appuie sur le tableau suivant selon les caractéristiques suivantes :

— Informations : Origine du Projet, Open Source, Langages implémentés, Gratuité.
— Support SCM - Source Code Manager (si oui lesquels)
— Gestion des builds
— Sécurité : Authentification d'utilisateurs, intégration LDAP (Lightweight Directory Access Protocol)
— Publication des résultats
— Interface Web

— Moteur de productions supportées
— Intégrations d'outil de suivi de projets et gestion de tickets (incidents, événements...)
— Bibliothèques de tests intégrées
— IDE intégrés
— Installation et configuration

Voici, page suivante, la veille technologique portant sur Jenkins, OpenMake Mojo, CruiseControl :

Information Projet	CruiseControl CruiseControl.NET	OpenMake Mojo	Hudson/Jenkins
Origine du projet	ThoughtWorks	OpenMake Software	java.net
Open Source	Oui	Non	Oui
Langages d'implémentation	Java	Java, C++, C#, C, JSP	Java
Gratuit	Oui	Oui	Oui
Support SCM	AccuRev AlienBrain ClearCase CMSynergy **CVS** **File system SCM** **Git** **HTTP file** MKS p4 PVCS StarTeam **Subversion** **Team Foundation Server** VSS	AccuRev ClearCase CA Harvest CM Synergy **CVS** **FileSystem SCM** **Git** Dimensions MKS p4 PVCS StarTeam **Subversion** **Team Foundation Server** VSS	AccuRev Bazaar/ BitKeeper ClearCase CM Synergy **CVS** **File system SCM** **Git** **HTTP file** Mercurial p4, PVCS StarTeam / Surround **Subversion** **Team Foundation Server** VSS
Gestion des builds	Oui	Oui	Oui
Parallel builds (abilité à build plusieurs projets simultanément)	Oui	Oui	Oui
Gestion de builds distribués	Oui	Oui	Oui
Gestion de SCM trigged ou poll based builds	Oui	Oui	Oui
Lancement de builds systématiques tous les xx temps	Oui	Oui	Oui
Reproduction de	Non	Oui	Oui

l'historique des builds			
Gestions de builds (promotions, déletions...)	Non	Oui.	Oui.
Detection et notification de tests ayant échoués.	?	Oui	Oui
	?	Oui	?
Sécurité –			
Authentification utilisateurs	Oui	Oui	Oui
Intégration LDAP	Non	Oui	Oui
Publication des résultats			
Email	Oui	Oui	Oui
FTP	Oui	Oui	Oui
Jabber	Oui	Non	Oui
RSS	Oui	Non	Oui
Interface WEB			
Ajouter, clôner, supprimer, modifier un projet	Non	Non	Oui
Mettre en pause un build	Oui	Oui	Oui
Accèder aux artifacts des builds	Oui	Non	Oui
Graphiques	Oui	Oui	Oui
Rafraîchissement	Oui	Non	Oui

automatique des pages			
Support de multi projets	Oui	Non	Oui
Moteurs de production supportés			
Shell/Command script	Oui	Oui	Oui
Ant	Oui	Oui	Oui
Groovy	Non	Non	Oui
OpenMake Meister	Oui	Oui	Oui
Maven	Oui	Oui	Oui
Maven2	Oui	Oui	Oui
Make	Non	Oui	Non
MsBuild	?	Oui	Oui
Nant	Oui	Oui	Oui
Rake (Ruby)	?	Non	Oui
Visual Studio	Non	Oui	Non
Intégration d'outils de suivi de projets et gestion de tickets			
Bugzilla	Non	Non	Oui
Jira	Non	Oui	Oui
Sourceforge.net	Non	Non	Non

Trac	Non	Non	Non
Bibliothèques de tests intégrées			
CppUnit	?	Oui	Oui
JUnit	Oui	Oui	Oui
NUnit	Oui	Oui	Oui
Installation et configuration			
Installateur Windows	Oui	Oui	Oui
Self Contained Distribution	Oui	Oui	Oui
Dépendances Additionelles	JRE, SCM client	JRE, Perl, SCM Client	JRE
Plateforme d'éxécution	JVM	Windows, AIX, Solaris, HP UX	JVM
Moteur de production préféré	Ant, Maven	Tous	Tous
Plateforme possible pour les projets	Java + tout ce qui est compilé à l'aide de Ant/Maven/Nant	Tous les languages !	Tous les languages !
Supports de multiple projets	Oui	Oui	Oui
Nécéssité de modifier les scripts de builds	Non	Non	Non
Fichiers texte de configuration	XML	?	XML

En conclusion de cette veille technologique portant sur les serveurs d'intégration continue, **il me semble logique de choisir Jenkins** (outil fork d'hudson suite à un différent entre son cofondateur et oracle) **pour ce projet de fin d'études.**

En effet, Jenkins, OpenMake Mojo, CruiseControl sont très complets et possède les mêmes capacités à quelques variations prêts. Simplement CruiseControl se base systématiquement sur des projets basés sur Maven/ANt/Nant. N'étant pas actuellement sûr de pouvoir faire tourner des projets c/c++ à l'aide de maven, il convient donc de l'abandonner.

Concernant les différences entre OpenMake Mojo et Jenkins, elles sont infimes. Mais Jenkins est plus simple d'installation, nécessite moins d'outils pour tourner (une simple JVM et un environnement JRE suffisent !). Ainsi, due à sa popularité, sa souplesse d'utilisation et son excellente intégration avec SonarQube et les logiciels de version de code subversion/git (voir la suite de ce rapport), Jenkins semble être l'outil le plus adapté.

3.2 Logiciels de gestions de versions

Un logiciel de gestion de versions (aussi appelés VCS en anglais pour Version Control System) est un logiciel permettant de stocker un ensemble de fichiers sous forme d'arborescence en conservant la chronologie de toutes les modifications qui ont été effectuées dessus. Cela va ainsi permettre de mutualiser le développement, notamment dans le cadre de gros projets.
Par exemple : *Deux développeurs travaillent sur une même source simultanément. Si le premier développeur travaillant sur la version encore non modifiée par le second soumet ses changements, le second aura ses changements à lui conservés.*

3.2.1 Git

Git est un logiciel de gestions de versions décentralisé et libre, crée par Linus Torvalds, créateur du noyau Linux.

FIGURE 3.1 – Git

Les avantages/inconvénients sont :
— + Pour ceux qui n'aiment pas SVN ou CVS
— + Bien plus rapide pour certaines tâches
— + Les opérations sur les branches ont un coût en temps faibles
— + L'historique total d'un projet sous forme d'arbre est proposé hors ligne
— + Système décentralisé
— - Apprentissage pouvant être compliqué
— - Pas optimal pour des utilisateurs seuls
— - Support windows limité

3.2.2 Mercurial

Mercurial est très proche de Git, puisqu'il est lui aussi un logiciel de gestions de versions décentralisées.
Les avantages/inconvénients sont :
— + Plus facile d'utilisation que Git

FIGURE 3.2 – Mercurial

— + Meilleure documentaion
— + Décentralisé
— - Impossible de merger deux parents

3.2.3 CVS

CVS (concurrent versions system) est le plus vieux VCS présent : Il a été créé aux alentours des années 80 ! Les avantages/inconvénients sont :

— + A été utilisé pendant de nombreuses années et est considéré comme "mature"
— - Parfois lent
— - Problème de sécurité
— - Ne supporte pas certaines actions

3.2.4 Subversion

Subversion a été crée et pensé comme une alternative à CVS pouvant résoudre des bugs de CVS en maintenant une haute compatibilité avec ce dernier.

FIGURE 3.3 – Subversion

Les avantages/inconvénients sont :

— + Système plus récent que CVS
— + Inclut des opérations atomic
— + Les opérations sur les branches ont un coût en temps faibles
— + Grande variété de plugins pour les IDEs
— + Non centralisé
— - Peut paraître plus lent
— - Quelques bugs encore présents

En conclusion, on pourrait choisir l'un des 3 logiciels de gestions de versions sans a priori, tant ils sont complets et faciles d'utilisation. Cependant, au sein de l'école Polytech Tours il existe déjà un serveur subversion fonctionnant pour l'ensemble de ces comptes. Il conviendra donc de s'interfacer avec, et par conséquent de choisir cet outil pour l'ensemble de ce projet.

3.3 Moteurs de Production

Les moteurs de production sont les logiciels dont la fonction principale consiste à automatiser l'ensemble des actions contribuant à produire un ensemble logiciel opérationnel à partir de données sources.

Les plus connus (et ceux qui nous intéressent dans le cadre de projets développés en C, C++, C#, Java) sont

— Make
— Nmake (Fournit par Microsoft et disponible dans Visual Studio)
— Cmake (Gère tout type de languages, C, C++, C#, Java...)
— Maven (Orienté Java mais gère tout type de languages.)
— Ant (Orienté Java mais gère tout type de languages.)
— SCons (Orienté C, C++, Fortran, support de Java, Qt, SWIG...)
— NAnt (Orienté C# et Visual Basic pour la plateforme .NET)
— Gradle (Orienté vers des projets Java, Scala, Groovy voir C++)
— GNU (AutoGen, Automake, Libtool, Make)
— etc...

La suite de cette veille technologique consistera à présenter et retenir les moteurs de production susceptibles de nous intéresser.

3.3.1 Maven/Ant

FIGURE 3.4 – Apache Maven

Apache Maven est un outil logiciel libre pour la gestion et l'automatisation de production des projets logiciels Java en général et Java EE en particulier. Il est semblable à l'outil **Ant** mais sa configuration est beaucoup plus simple ! C'est pourquoi nous préférerons dans ce projet l'utilisation de Maven.

L'avantage de maven est sa capacité à s'allier avec Jenkins afin de simplifier la partie de création de Jobs sur la plateforme Jenkins. Ainsi le but dans ce projet, sera d'utiliser au maximum maven (si possible pour les projets C, C++, C# , grâce à l'utilisation de plugins) afin de simplifier au maximum la tâche de l'utilisateur lors de la configuration des ressources pour faire tourner jenkins, sonar et nexus.

Repository Manager, Nexus de Sonatype

Un élément clé de Maven est son aptitude à fonctionner en réseau. C'est là qu'interviennent nos repository manager dont l'utilité est double :

1. Agit comme un proxy configurable entre les repositories publics maven et l'environnement utilisateur : Cela procure de nombreux avantages (accélération du processus de build, garder le contrôle de ce qui est téléchargé, des licences des dépendances utilisées... De plus, si l'internet est coupé (ce qui arrive fréquemment les jours de pfe entre 10h30 et 12h00...), cela n'impacte pas les développeurs !).
2. Permet d'organiser l'endroit où les déploiements des projets mavens se feront.

On peut considérer qu'ils sont l'équivalent de nos outils de gestion de code source SVN et GIT mais appliqué ici aux artefacts et dépendances maven générés lors des builds.

Pour appuyer un peu plus nos propos, citons le président d'Artifact Software : Ron Wheeler :

> *"If you are starting out with Maven, start using a repository manager. A repo is essential to becoming a happy Maven user." Ron Wheeler, President, Artifact Software*

FIGURE 3.5 – Nexus Sonatype

Le plus connu et le plus utilisé de ces repository manager est Nexus de Sonatype, que nous utiliserons dans la suite de ce projet. En effet, en plus d'être libre, il est doté d'une forte communauté et d'une excellente documentation. Il est simple d'installation et complet.

3.3.2 Autres moteurs de productions

Cependant dans le cadre des projets développés dans un autre langage que Java, il est possible que Maven ne suffise pas (ou que l'utilisation de maven-nar-plugin pouvant être théoriquement utilisée pour faire tourner des projets en c++/c, c# ne fonctionne pas). Dans ce cas, un autre moteur de production devra être utilisé. Tous sont supportés par Jenkins et SonarQube, l'utilisateur aura donc le choix : MSbuild, C/C++, ... Comme Jenkins et SonarQube peuvent tout de même travailler sur des projets C, C++, C# il serait fort dommage de passer à côté ! Il faudra simplement indiquer que la partie configuration sur Jenkins et SonarQube sera alors plus compliquée.

3.4 Système de suivi de tickets et gestion de projets

Il existe également dans le monde de l'intégration continue des systèmes complets dédiés au suivi de bugs, à la gestion des incidents... Ces systèmes, dont les plus connus sont **Trac** et **Jira**, sont très puissants et complets et en général Open Source. Il serait donc possible de les mettre en place dans le cadre de ce projet de fin d'études. Cependant, il convient de faire des choix quant aux nombres d'outils mis en place et pour lesquels des formations devront être prévues. Due à leur complexité (ce sont des outils extrêmement complets), **j'ai donc décidé de ne pas les mettre en place sur ce projet.**

Qualité de Code

4.1 Outils disponibles sur le marché

Il existe une multitude d'outils disponibles pour mesurer la qualité de code, souvent centrés sur un seul langage et possédant chacun ses propres spécificités (le domaine de la qualité de code est vaste et repose sur de nombreux critères...) :

1. CodePro Analytix
2. **PMD**
3. **FindBugs**
4. Cobertura
5. Emma
6. **Checkstyle**
7. JBoss Tattletale
8. UCDDetector
9. QALab
10. XRadar
11. Clirr
12. JDiff
13. JLint
14. JDepend
15. cloc
16. etc... La liste est très longue !

Cependant, pour des raisons pratiques, on ne peut implémenter tous ces outils. Il serait en effet inutile de penser pouvoir former l'utilisateur final à toutes ces technologies ! Il nous fallait ainsi trouver un outil pouvant regrouper si possible les précédents outils et proposer un support complet pour des projets pouvant énormément différer (différences de langages, différences d'outils d'intégration continue...).

4.1.1 SonarQube

Il n' existe à ce jour un seul outil capable de réaliser l'ensemble des conditions vues précédemment : SonarQube (anciennement Sonar...).

On peut au premier abord penser que Sonar est moins utilisé que ses confrères. Cependant, Sonar se base sur les outils PMD, Findbugs, Checkstyle et Squid. Par raccourci, utiliser Sonar revient à utiliser les technologies les plus utilisées actuellement sur le marché ce qui est un avantage non négligeable.

Les avantages de Sonar sont :

— Il supporte notamment plus de 25 langages, ce qui suffit amplement dans le cadre de la mise en route d'un serveur pour l'école Polytech Tours.
— Il se combine avec Maven ou Ant ou Gradle (voir la partie Moteur de Production).
— Il possède une forte communauté.

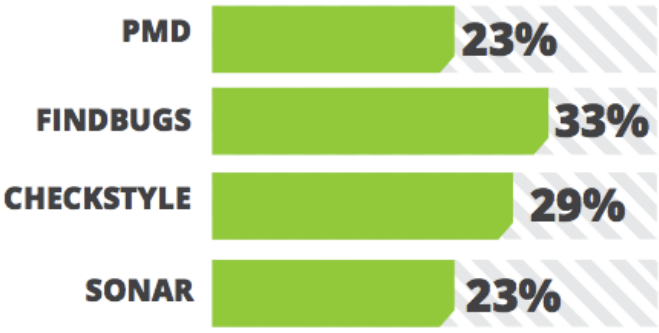

FIGURE 4.1 – Utilisation de Sonar parmi les outils de qualité de code les plus utilisés

— Il offre un support complet d'outils de qualité de code.
— Il est simple d'utilisation !

Il est donc logique, pour l'ensemble des raisons citées ci-dessus, de choisir SonarQube comme unique outil de Qualité de Code.

Nous verrons dans le cahier de spécifications de quelle manière nous allons intégrer Sonar à notre solution finale.

4.1.2 Plugins choisis

Il est possible pour qui le souhaite d'étendre le coeur de Sonar afin d'augmenter les fonctionnalités de Sonar.

Les plugins sont actuellement divisés en plusieurs catégories :

— **Métriques Additionnelles** : permet de calculer de nouvelles métriques pour Sonar. Il s'agit en général de permettre le support d'outils externes (JaCoCo, JIRA, JMeter, Mantis, SonarJ, etc.).
— **Nouveaux Languages** : par défaut, Sonar ne gère que le langage Java, mais grâce à l'ajout de plugins, de nombreux autres langages peuvent être gérés (C, C#, en ce qui nous concerne).
— **Visualisation** : plugins permettant un affichage différent des métriques calculées par Sonar.
— **Gouvernance** : offre le support de nouvelles méthodologies d'analyse, comme par exemple SQALE.
— **Intégration** : permet une meilleure intégration de l'outil Sonar dans l'usine logicielle, par exemple en offrant un support avec Hudson ou Bamboo (serveurs d'intégration continue).
— **IDE** : plugins destinés aux IDE (outil de développement comme Eclipse ou IntelliJ IDEA) offrant une intégration plus poussée avec Sonar.

On peut retrouver l'ensemble des plugins à l'addresse :Plugin Library ainsi qu'une documentation plus complète.

Voici la liste des plugins Sonar, leur utilité, et la décision quant à leur choix final :

Nom du plugin	Type	Description	Payant	Décision
ABAP	Languages		Oui	Non
Android	Languages		Non	Non
C/C++	Languages		Oui	**A voir**
C#	Languages		Non	**Oui**
Cobol	Languages		Oui	Non
Delphi/Pascal	Languages		Non	Non
Drools	Languages		Non	Non
Erlang	Languages		Non	Non
Flex/ActionScript	Languages		Non	Non
Groovy	Languages		Non	Non
Java	Languages		Non	**Oui**
JavaScript	Languages		Non	Non
Natural	Languages		Oui	Non
Pacbase	Languages		Oui	Non
PHP	Languages		Non	Non
PL/I	Languages		Oui	Non
PL/SQL	Languages		Oui	Non
Python	Languages		Non	Non
VB.NET	Languages		Oui	Non
Visual Basic 6	Languages		Oui	Non
WEB	Languages		Non	Non
XML	Languages		Non	Non
Developer Cockpit	Developer Tools	Permet à chaque developer d'identifier leur contributions individuelles à un projet et favorise de meilleures pratiques dans l'autogestion de qualité de code	Oui	Non
Eclipse	Developer Tools	Permet de voir directement dans Eclipse les défauts rassemblés par SonarQube.	Non	**Oui**
Issues Report	Developer Tools	Lance des analyses locales sur le code source (compatibles tout languages).		
Views	Governance	Portfolio Management : Les projets peuvent être regroupés en applications, les applications en équipes, équipes en département...	Oui	Non
PDF Report	Governance	Génére des rapports PDF depuis l'analyse du projet (**projets en Java**).	Non	**Oui**
SQALE	Governance	Ajoute une implémentation de SQALE Methodology pour pallier la dette technique.	Oui	Non
Branding	Integration	Ajoute son propre logo à l'interface utilisateur de SonarQube	Non	**Oui**
Build Breaker	Integration	Si les alertes seuil prédéfinies sont atteintes, l'analyse échoue.	Non	Non
Fortify	Integration	Importe les rapports de Fortify (Security Rating, number of issues, and vulnerability issues).	Non	Non
Google Analytics	Integration	Ajoute un script de traquage Google Analytics à l'application SonarQube	Non	Non
Google Calendar	Integration	Poste un évènement quand un projet est analysé par SonarQube.	Non	Non
Hudson	Integration	Permet de lancer des analyses sonarqube depuis Hudson.	Non	Non
Jenkins	Integration	Permet de lancer des analyses sonarqube depuis Jenkins.	Non	**Oui**
Maven Report	Integration	Ajoute un lien au site Maven pour referencer le projet sonarqube.	Non	**Oui**
Piwik	Integration	Permet l'utilisation d'une instance SonarQube avec un serveur piwik (alternative à google analytics).	Non	Non
SCM Activity	Integration	Collectionne des informations SCM et affiche la date du commit et l'identité de la personne ayant committé à chaque ligne de code.	Non	**Oui**
Crowd	Authentification& Authorization	Permet d'autoriser la delegation de l'authentification SonarQUbe à Atlassian Crowd	Non	Non
LDAP	Authentification&	Permet d'autoriser la delegation de l'authentification et autorisation SonarQube à **LDAP et Microsoft Active**	Non	**Oui**

	Authorization	**Directory**		
OpenID	Authentification& Authorization	Permet d'autoriser la delegation de l'authentification SonarQube à un fournisseur OpenId.	Non	Non
PAM	Authentification& Authorization	Permet d'autoriser la delegation de l'authentification SonarQube à un sous système PAM.	Non	Non
Abacus	Additional Metrics	Estime la complexité de chaque fichier du projet grâce à la méthode abacus.	Non	**Oui**
Artifact Size	Additional Metrics	Donne la taille de l' artifact généré par le projet	Non	**Oui**
Build Stability	Additional Metrics	Génère des rapports basée sur les informations issues des builds d'intégration continue.	Non	**Oui**
Jira Issues	Additional Metrics	Retourne le nombre de publications de projet de JIRA et permet la création de publications de projet de JIRA.	Non	Non
JMeter	Additional Metrics	Donne les résultats du test Jmeter à SonarQube.	Non	Non
Mantis	Additional Metrics	Retourne le nombre de publications d'un projet développé avec l'outil Mantis Bug Tracker	Non	Non
Security Rules	Additional Metrics	Permet de définir et gérer un ensemble de règles.	Non	**Oui**
Taglist	Additional Metrics	Offre des spécialités sur les règles définies par Squid (*par exemple des tags comme //TODO*)	Non	**Oui**
Thucydies	Additional Metrics	Propose des fonctionnalités sur les tests basés sur Thucydides	Non	Non
Trac	Additional Metrics	Utilise TracXML-RPC plugin pour se connecter à une instance de Trac et afficher des metriques sur les tickets ouverts.	Non	Non
Useless Code Tracker Plugin	Additional Metrics	L'objectif de ce plugin est de dire quel nombre de ligne peut être réduit dans l'application.	Non	**Oui**
Violation Density	Additional Metrics	Calcule une nouvelle densité de violation (métrique).	Non	**Oui**
Doxygen	Vizualization Reporting	Ce plugin génére la documentation de l'application en se basant sur Doxygen et Graphviz	Non	**Oui**
Motion Chart	Vizualization Reporting	Ce plugin offre de widgets affichant l'évolution à travers le temps de 4 métriques choisies.	Non	**Oui**
SCM Stats	Vizualization Reporting	Ce plugin calculre et nourrit SonarQube avec 4 nouvelles métriques : Commits/Author, Commits/Clock Hour, Commits/Week Day, Commits/Month	Non	**Oui**
Tab Metrics	Vizualization Reporting	Ajoute une nouvelle perspective (contenant des informations sur toutes les métriques stockée dans la base de données de Sonar) dans l'afficheur de composants (component viewer).	Non	**Oui**
Timeline	Vizualization Reporting	Affiche l'historique dans un widget utilisant Google Vizualisation Annotated TimeLine component	Non	Non
Widget Lab	Vizualization Reporting	Offre de nouveaux widgets WMR rules compliance, manual severity reviews, comments and documentation, duplications...	Non	**Oui**
Localization	Localization	Différents package de langues.	Non	**Oui**

Un seul doute subsiste. En effet, il est nécessaire d'utiliser le plugin C/C++ afin de pouvoir utiliser SonarQube dans le cadre de projets PIL développés en C/C++.
Or ce plugin s'avère être disponible en version payante (voir Plugin Library). D'après les dires de Monsieur TKind't il existe également une version gratuite.
Sans cette version gratuite, on ne pourra pas construire de projets basés sur le langage C/C++ (et ce sera aussi le cas pour des projets .NET et construits sous visual studio).

C'est l'un des plus gros risques sur ce projet, la gestion des projets c/c++ et c#.

Conclusion

Ce document retrace l'ensemble des solutions choisies dans le cadre de projet de fin d'études : "Mise en place d'un serveur d'intégration continue" et notamment les raisons de ces choix. La solution se base ainsi sur les outils : **Maven, Nexus, Jenkins, Sonarqube**. Leur fonctionnement, leur mise en place et leur interfaçage entre l'ensemble de ces outils est relaté dans l'ensemble du document cahier de spécifications.

Mise en place d'un serveur d'intégration continue et qualité de code

Département Informatique
5e année
2013 - 2014

Veille Technologique

Résumé : Document représentant la veille technologique effectuée par l'élève Anne Castier pour son projet de fin d'études "Mise en place d'un serveur d'Intégration Continue et Qualité de Code."

Mots clefs : Serveur, Intégration Continue, Qualité de Code

Abstract: Document about the technology watch made by Anne Castier for her final project " Server of Continuous Integration and Quality of Code."

Keywords: Server, Continuous Integration, Quality Of Code

Encadrants
Vincent T'Kindt
vincent.tkindt@univ-tours.fr
Nicolas Ragot
nicolas.ragot@univ-tours.fr

Étudiants
Anne CASTIER
anne.castier@etu.univ-tours.fr

DI5 2013 - 2014

Université François-Rabelais, Tours

École Polytechnique de l'Université de Tours
64, Avenue Jean Portalis
37200 TOURS, FRANCE
Tél. +33 (0)2 47 36 14 14
www.polytech.univ-tours.fr

Département Informatique
5e année
2008 - 2009

Manuel Installation

Serveur Intégration Continue et Qualité de Code

Encadrants
Nicolas Ragot
nicolas.monmarche@univ-tours.fr
Vincent T'Kindt
tkindt@univ-tours.fr

Étudiants
Anne CASTIER
anne.castier@etu.univ-tours.fr

DI5 2013 - 2014

Université François-Rabelais, Tours

Version du 16 avril 2014

Table des matières

Introduction

Ce manuel relate l'installation de départ du serveur d'intégration continue, suite aux choix effectués durant la veille technologique de ce projet de fin d'études. Cette installation se fait sur une machine virtuelle fonctionnant sous Windows 7. **Le choix de ce système d'exploitation est lié à la volonté de pouvoir travailler avec des environnements microsoft comme Visual Studio et le langage C#.** *Cependant, on peut tout à fait adapter les instructions de ce manuel pour réinstaller le serveur sous Ubuntu, pour peu que les contraintes liées à l'utilisation d'environnement de travail Microsoft disparaissent.* On y détaillera ainsi l'installation des prérequis, ainsi que nos deux outils principaux Jenkins et SonarQube.

Pré-installations nécessaires

L'utilisation de Jenkins et SonarQube requiert notamment l'installation d'un JDK récent et de Maven.

2.1 Installation java 1.7

Voici le pas à pas de l'installation du jdk 1.7.
Rendez vous d'abord sur la page suivante : *http ://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html*

Java SE Development Kit 7u51

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

Thank you for accepting the Oracle Binary Code License Agreement for Java SE; you may now download this software.

Product / File Description	File Size	Download
Linux ARM v6/v7 Hard Float ABI	67.7 MB	jdk-7u51-linux-arm-vfp-hflt.tar.gz
Linux ARM v6/v7 Soft Float ABI	67.68 MB	jdk-7u51-linux-arm-vfp-sflt.tar.gz
Linux x86	115.65 MB	jdk-7u51-linux-i586.rpm
Linux x86	132.98 MB	jdk-7u51-linux-i586.tar.gz
Linux x64	116.96 MB	jdk-7u51-linux-x64.rpm
Linux x64	131.8 MB	jdk-7u51-linux-x64.tar.gz
Mac OS X x64	179.49 MB	jdk-7u51-macosx-x64.dmg
Solaris x86 (SVR4 package)	140.02 MB	jdk-7u51-solaris-i586.tar.Z
Solaris x86	95.13 MB	jdk-7u51-solaris-i586.tar.gz
Solaris x64 (SVR4 package)	24.53 MB	jdk-7u51-solaris-x64.tar.Z
Solaris x64	16.28 MB	jdk-7u51-solaris-x64.tar.gz
Solaris SPARC (SVR4 package)	139.39 MB	jdk-7u51-solaris-sparc.tar.Z
Solaris SPARC	98.19 MB	jdk-7u51-solaris-sparc.tar.gz
Solaris SPARC 64-bit (SVR4 package)	23.94 MB	jdk-7u51-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	18.33 MB	jdk-7u51-solaris-sparcv9.tar.gz
Windows x86	123.64 MB	jdk-7u51-windows-i586.exe
Windows x64	125.46 MB	jdk-7u51-windows-x64.exe

FIGURE 2.1 – JDK - Java

Acceptez les conditions et installez la version **Windows x86.**
Exécutez ensuite l'éxécutable, suivez l'installation. Laissez le repertoire d'installation par défaut (à savoir C :/Program Files/Java/jdk.1.7.0_51). .

Ajoutez une **variable d'environnement utilisateur** JAVA_HOME pointant sur C :/Program Files/Java/jdk.1.7.0_51

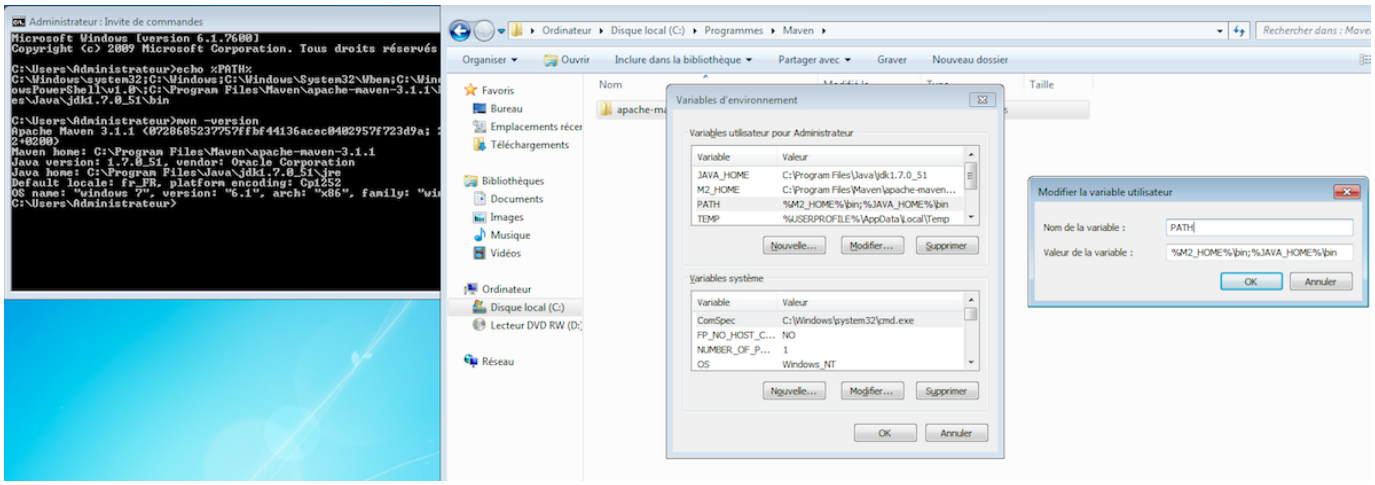

FIGURE 2.2 – Variables JDK - Java

De plus, il est necéssaire d'aller dans le **panneau de configuration windows**, onglet **Java**. Décochez le jre actuel, et ajoutez celui installé : Cliquez sur l'onglet recherche et allez chercher le dossier jdk installé.

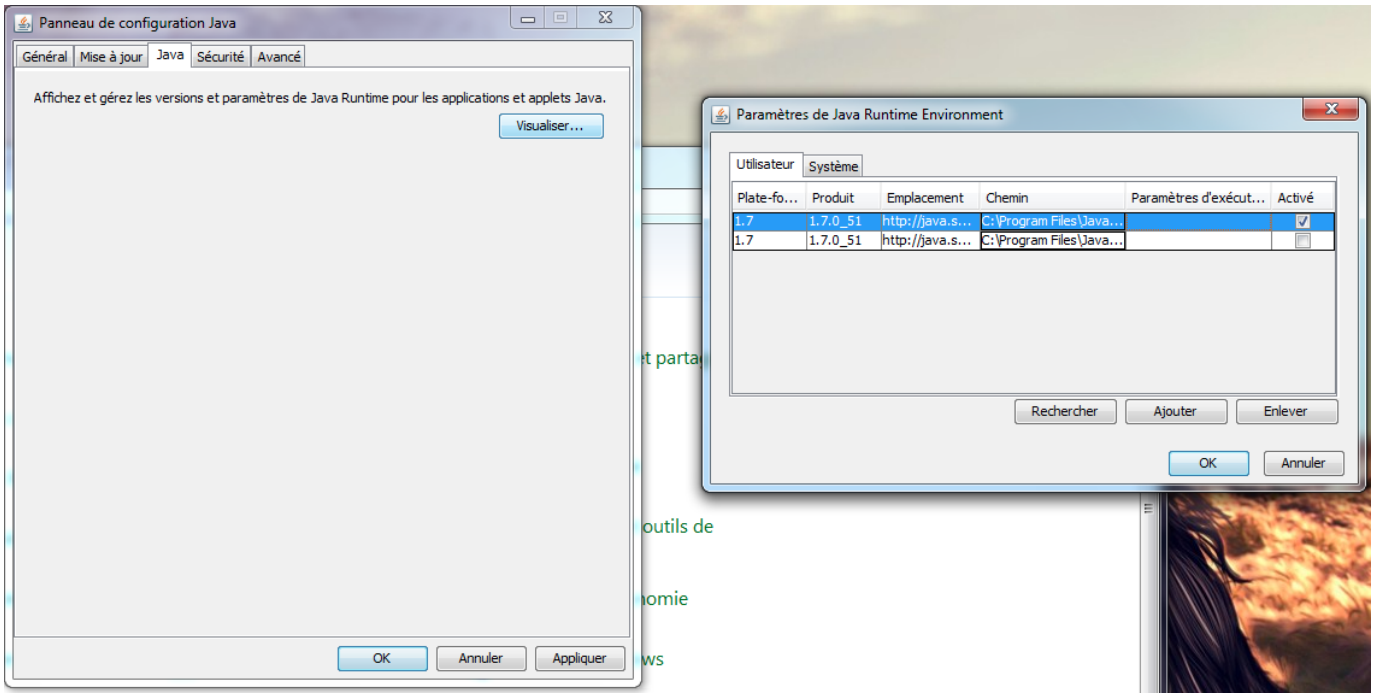

FIGURE 2.3 – JRE Configuration - Java

.

Voilà, l'installation du JDK est finie ! Si tout s'est bien passé, lancez une invite de commandes (un terminal...), et éxécutez la commande *java -version*. Le résultat affiché devrait être notre version de Java (la 1.7.51). On peut à présent passer à celle de Maven, autre incontournable pour notre serveur.

2.2 Installation Maven

L'installation de maven est très rapide.
Il faut tout d'abord télécharger une version récente de maven à l'adresse suivante :
http ://maven.apache.org/download.cgi (**Choisissez bien la version bin.**)

Maven 3.2.1

This is the current stable version of Maven.

	Link	Checksum	Signature
Maven 3.2.1 (Binary tar.gz)	apache-maven-3.2.1-bin.tar.gz	apache-maven-3.2.1-bin.tar.gz.md5	apache-maven-3.2.1-bin.tar.gz.asc
Maven 3.2.1 (Binary zip)	apache-maven-3.2.1-bin.zip	apache-maven-3.2.1-bin.zip.md5	apache-maven-3.2.1-bin.zip.asc
Maven 3.2.1 (Source tar.gz)	apache-maven-3.2.1-src.tar.gz	apache-maven-3.2.1-src.tar.gz.md5	apache-maven-3.2.1-src.tar.gz.asc
Maven 3.2.1 (Source zip)	apache-maven-3.2.1-src.zip	apache-maven-3.2.1-src.zip.md5	apache-maven-3.2.1-src.zip.asc
Release Notes	3.2.1		
Release Reference Documentation	3.2.1		

FIGURE 2.4 – Maven Installation

Dézippez vers **C :/Program Files/Java/Program Files/Maven** (Vous devrez créer le dossier). Ajoutez une variable d'environnement utilisateur **M2_HOME** pointant sur C :/Program Files/Maven/apache-maven-3.1.1 (cf screenshots). Ajoutez aussi une variable d'environnement utilisateur **PATH** contenant %M2_HOME%/bin ;%JAVA_HOME%/bin

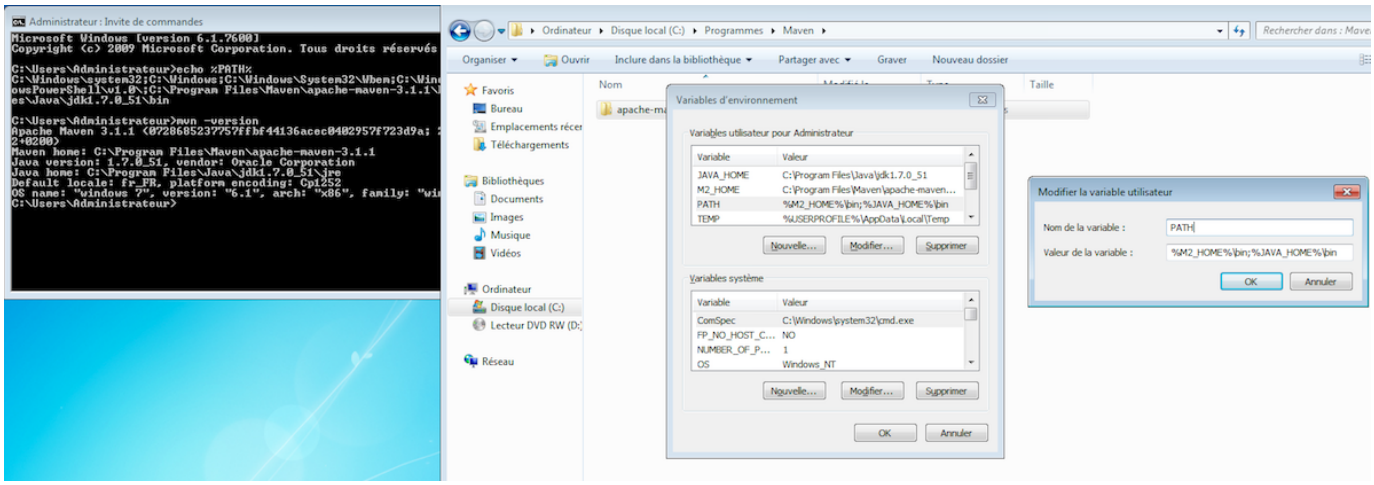

FIGURE 2.5 – Variables JDK - Java

Vérifiez la bonne installation à l'aide de la commande *mvn –version* dans un terminal (invite de commandes). Vient ensuite l'étape de configuration : Configurez le fichier settings.xml (dans le dossier conf) du dossier Maven.
Remplacez par :
<settings xmlns="http ://maven.apache.org/SETTINGS/1.0.0"
xmlns :xsi="http ://www.w3.org/2001/XMLSchema-instance"
xsi :schemaLocation="http ://maven.apache.org/SETTINGS/1.0.0
http ://maven.apache.org/xsd/settings-1.0.0.xsd">
<localRepository/>

<interactiveMode/>
<usePluginRegistry/>
<offline/>
<pluginGroups/>
<servers/>
<mirrors/>
<proxies/>
<profiles/>
<activeProfiles/>
</settings>

2.3 Installation Tomcat7

Nous avons décidé, suite à la veille technologique, d'utiliser Tomcat7 comme conteneur de nos 2 outils principaux SonarQube et Jenkins. Cette partie va détailler l'installation et la configuration (relativement minime) de Tomcat7.

2.3.1 Installation

Tout d'abord, rendez-vous sur le site officiel *http ://tomcat.apache.org/download-70.cgi*

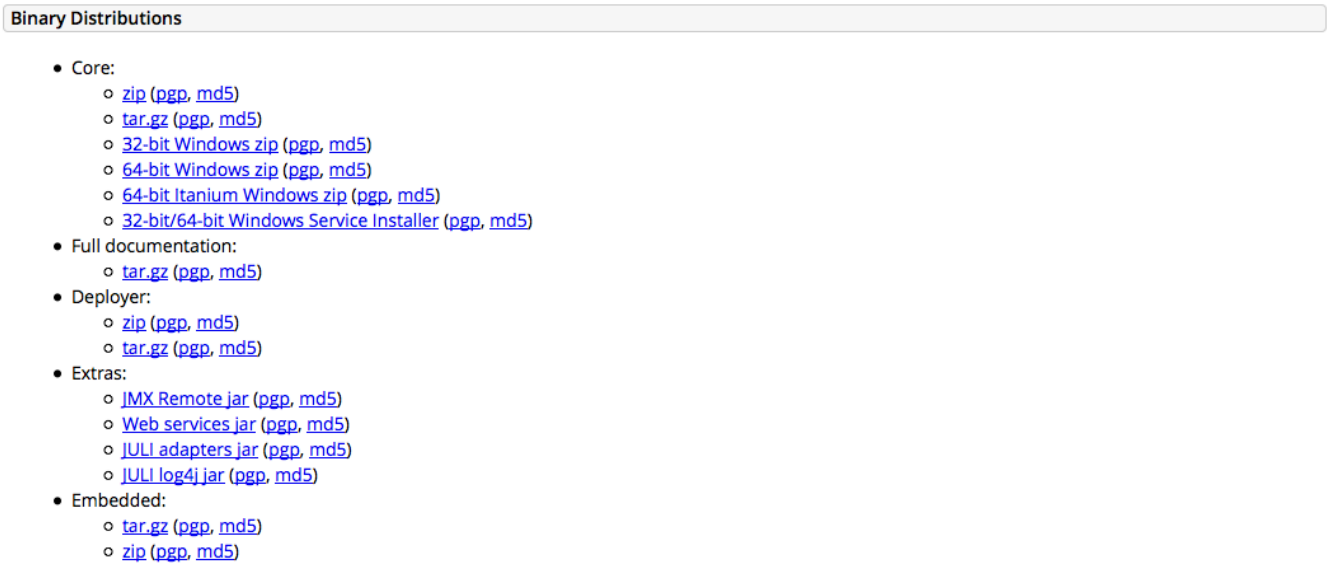

FIGURE 2.6 – Installation Tomcat

Téléchargez l'installateur windows (version 32 bits). Double-cliquez sur le .exe et suivez l'installation telle qu'elle est décrite par les screenshots suivants :

.

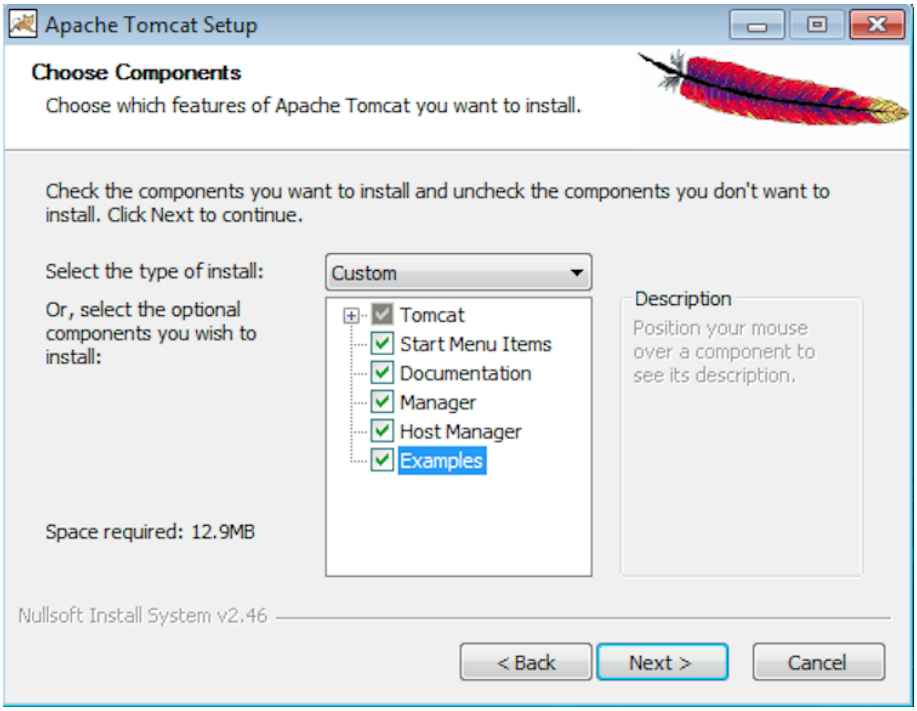

FIGURE 2.7 – Installation Tomcat

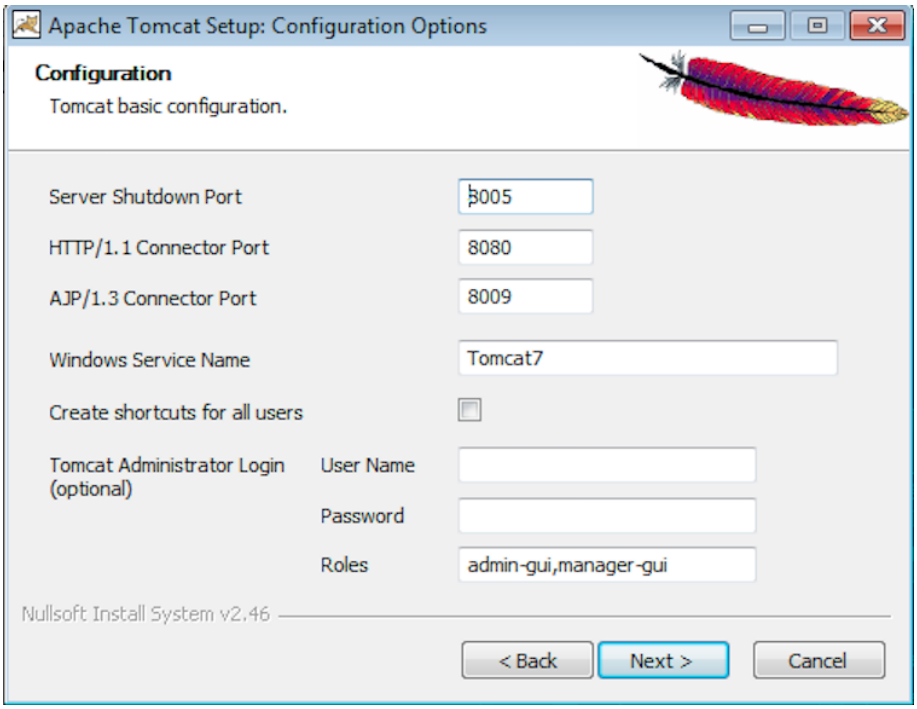

FIGURE 2.8 – Installation Tomcat

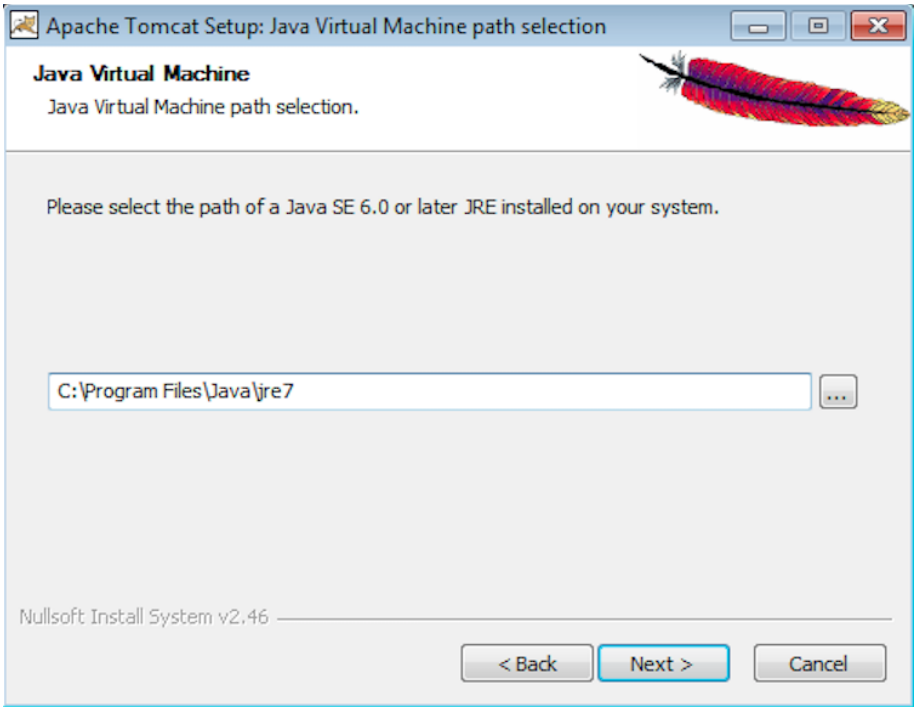

FIGURE 2.9 – Installation Tomcat

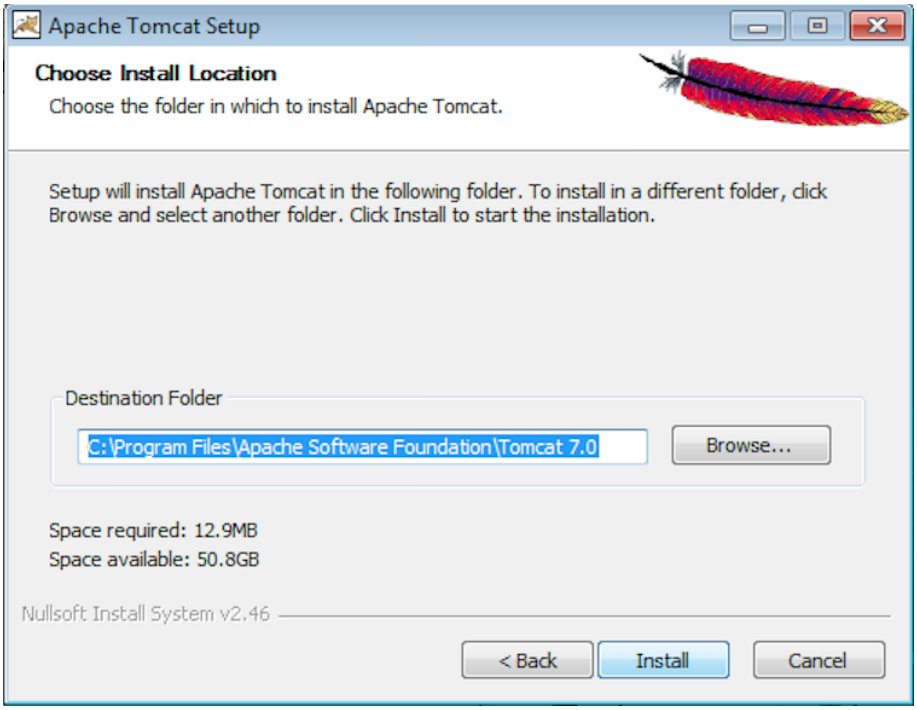

FIGURE 2.10 – Installation Tomcat

On va de plus configurer de plus Tomcat. On lance l'éxécutable **Tomcatw**, puis on se rend dans l'onglet **Java**, on rajoute les options suivantes :
-Djava.awt.headless=true
-Xmx128m
-XX :+UseConcMarkSweepGC

-XX :+CMSIncrementalMode
-XX :+CMSClassUnloadingEnabled
-XX :+CMSPermGenSweepingEnabled
-XX :MaxPermSize=128m
-Dsvnkit.http.sslProtocols="SSLv3"

Mais, également, dans le fichier *server* du dossier conf (de Tomcat), on va modifier la ligne suivante :
<!– Define an AJP 1.3 Connector on port 8009 –>
<Connector port="8009" protocol="AJP/1.3" redirectPort="8443"
enableLookups="false" maxThreads="50" minSpareThreads="5" maxSpareThreads="20"
acceptCount="100" connectionTimeout="2000" />
et rajoutez : *URIEncoding="UTF-8"* à *<Connector port="8080" ..... >* Cela permettra d'afficher les caractères spéciaux.
Lancez le serveur à la fin de l'installation : **L'utilisation de l'application Tomcat7w dans le dossier bin de Tomcat 7.0, permet de gérer l'arrêt et le démarrage de Tomcat.**

Si tout se passe bien vous devriez pouvoir contacter le serveur apache à l'adresse *localhost :8080* comme le montre le screenshot suivant :

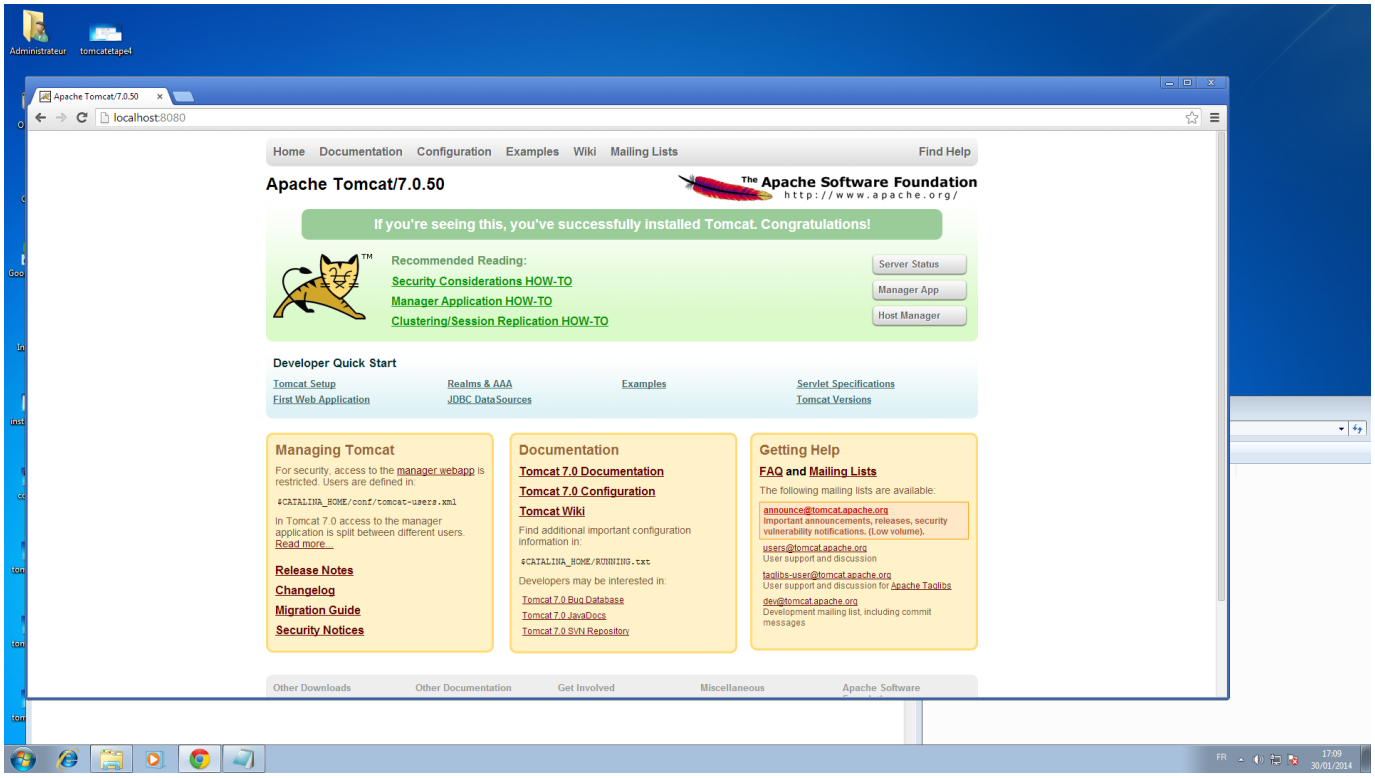

FIGURE 2.11 – Installation Tomcat

Installation Jenkins

Une fois les prérequis installés et configurés, on peut passer à l'installation de notre premier outil indispensable à notre serveur d'intégration continue : "Jenkins".

3.1 Installation Jenkins en tant que webapps

1. Commencez tout d'abord par arrêter tomcat (grâce à l'application **Tomcatw** vue précédemment).
2. Téléchargez la version .war de Jenkins (**onglet Latest and Greatest**) sur le site officiel de Jenkins : *http ://jenkins-ci.org/*

FIGURE 3.1 – Installation Jenkins

3. Copiez/Collez ce fichier .war téléchargé dans le dossier webapps de tomcat
4. Ensuite, vous pouvez relancez Tomcat. Pour vérifier que tout s'est bien passé, allez sur localhost :8080/jenkins.

3.2 Plugins

L'un des grands avantages de Jenkins, est bien évidemment sa capacité à s'étendre et s'adapter aux besoins de chacuns grâce à une liste de plugins impressionante (plus de 600 !).
La liste ci-dessous est exhaustive et a été décidée suite à la réunion tenue courant janvier à la fin de la veille

technologique. Bien sûr il est possible d'en ajouter autant que l'on souhaite et selon les besoins d'évolutions de Jenkins.

Pour ajouter des plugins à Jenkins, il suffit de se rendre (avec un compte administrateur) sur la page d'acceuil de Jenkins. On se rend d'abord dans l'onglet **Administrer Jenkins**, puis sur **Gestion des Plugins** :

Dans l'onglet **Disponibles**, cochez l'ensemble des plugins à installer :

— Xunit
— MSBuild, MSTest, MsTestRunner
— Sonar
— Nant
— Cmake
— Maven Project
— M2 Release
— Subversion
— Active Directory, LDAP
— Extended Read Permission Plugin
— Create Job Advanced

Redémarrez ensuite Tomcat comme demandé.

3.3 Configuration

Pour la bonne utilisation de Jenkins, il est nécessaire de le configurer et de spécifier le chemin des outils qu'il aura à utiliser (Maven, JDK, compilateurs MSBuild...). Rendez vous sur l'adresse de votre serveur Jenkins : (Pour nous, *http ://localhost :8080/jenkins*). Allez dans l'onglet **Administrer Jenkins**, onglet **Configuration Générale**

Configurez votre serveur à l'instar des screenshots suivants (à compléter) : (Note, certaines de ces installations requises vont être installées par la suite. Vous devrez ainsi revenir plus tard sur cette partie pour rajouter les éléments manquants : SonarQube, MSbuild etc.).

FIGURE 3.2 – Installation Jenkins

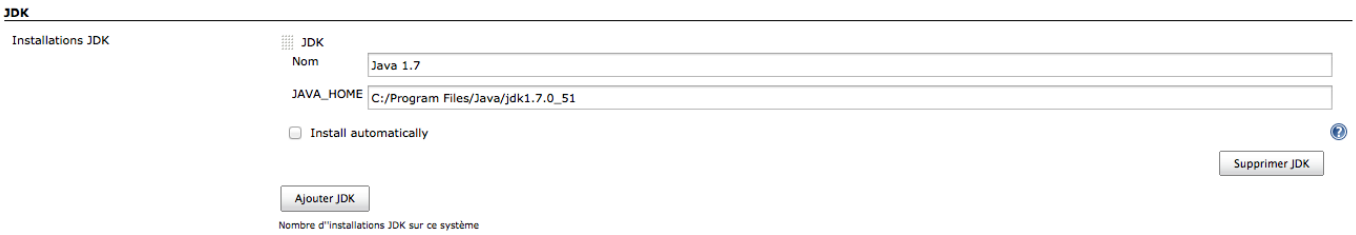

FIGURE 3.3 – Installation Jenkins

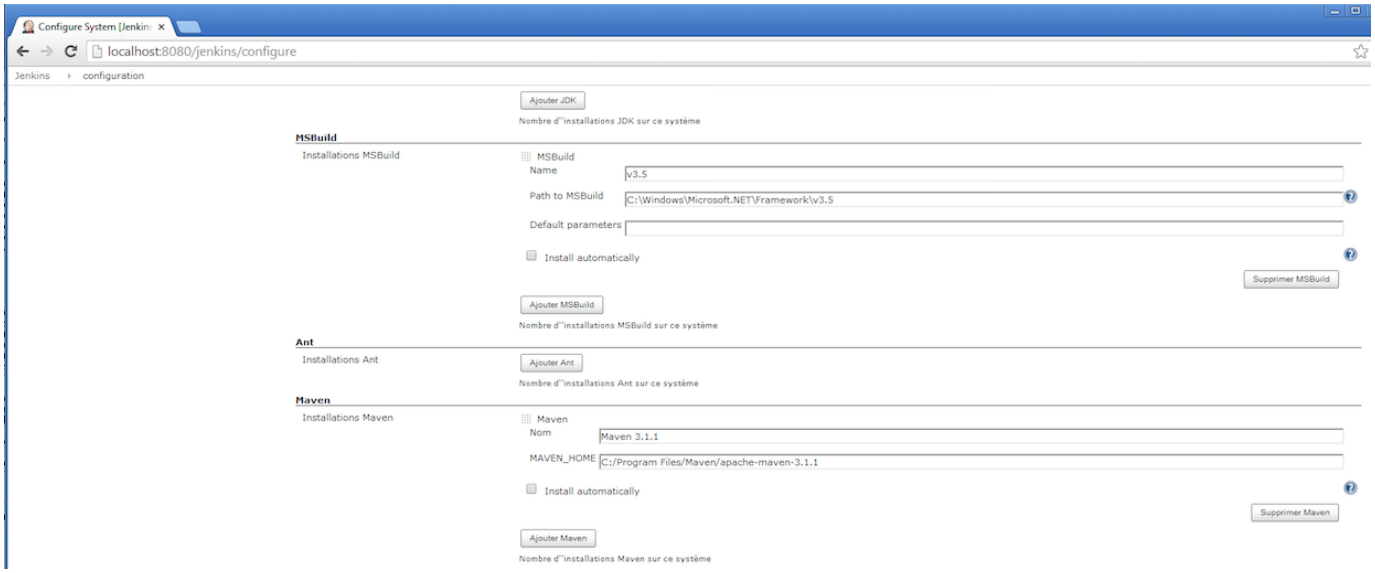

FIGURE 3.4 – Installation Jenkins

.

.

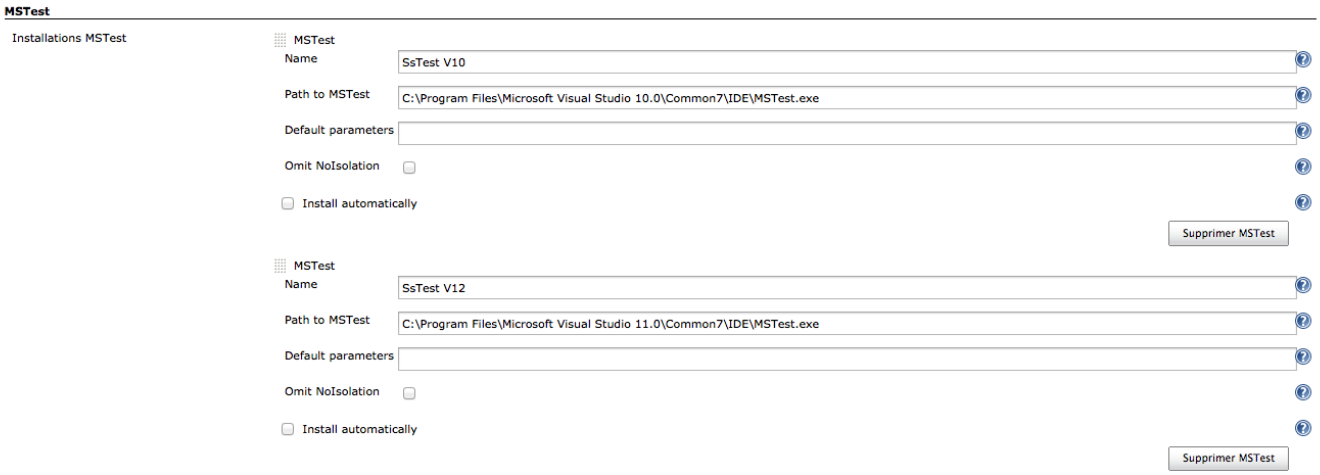

FIGURE 3.5 – Installation Jenkins

.

.

.

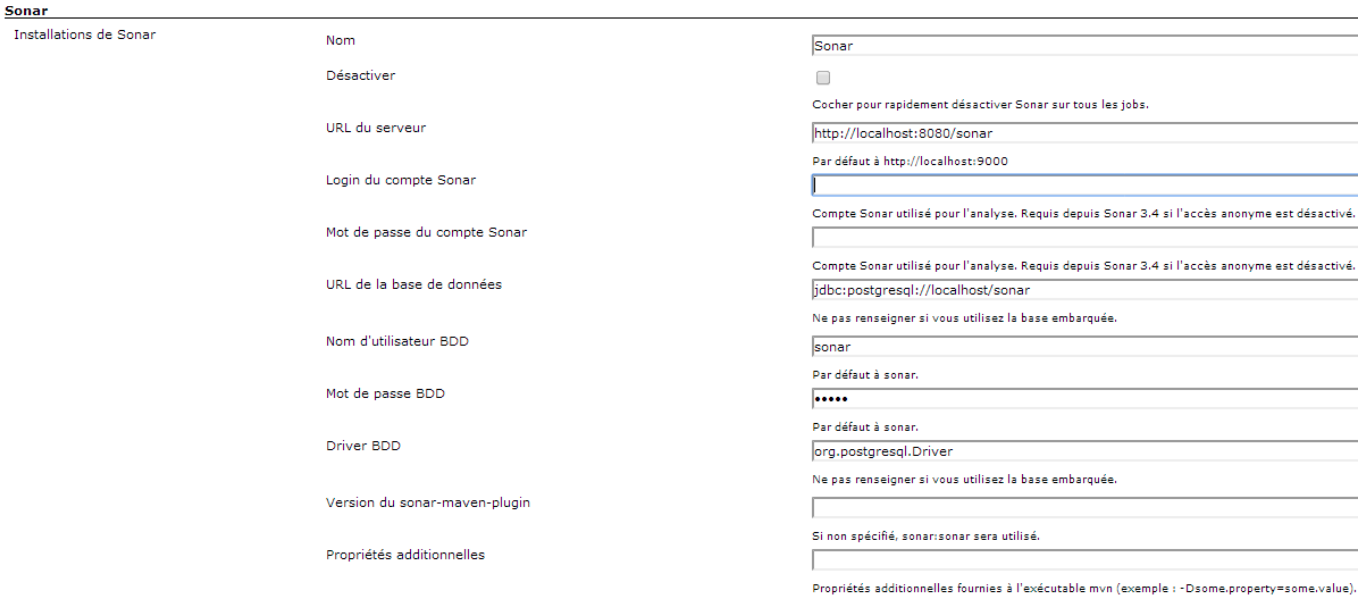

FIGURE 3.6 – Installation Jenkins

.

FIGURE 3.7 – Installation Jenkins

.

.

3.4 Sécurité basique

Cette partie permet l'utilisation de l'active directory pour permettre à tous les étudiants de se logguer en utilisant leurs identifiants étudiants.

Pour cela, rendez vous dans l'interface web de Jenkins, onglet **Administrer Jenkins** , onglet **Configurer la sécurité globale**. Cochez la case **"Activez la sécurité"**, puis dans **Royaume pour la sécurité (Realm)** cochez **LDAP**. Configurez cette partie comme le montre le screenshot suivant :

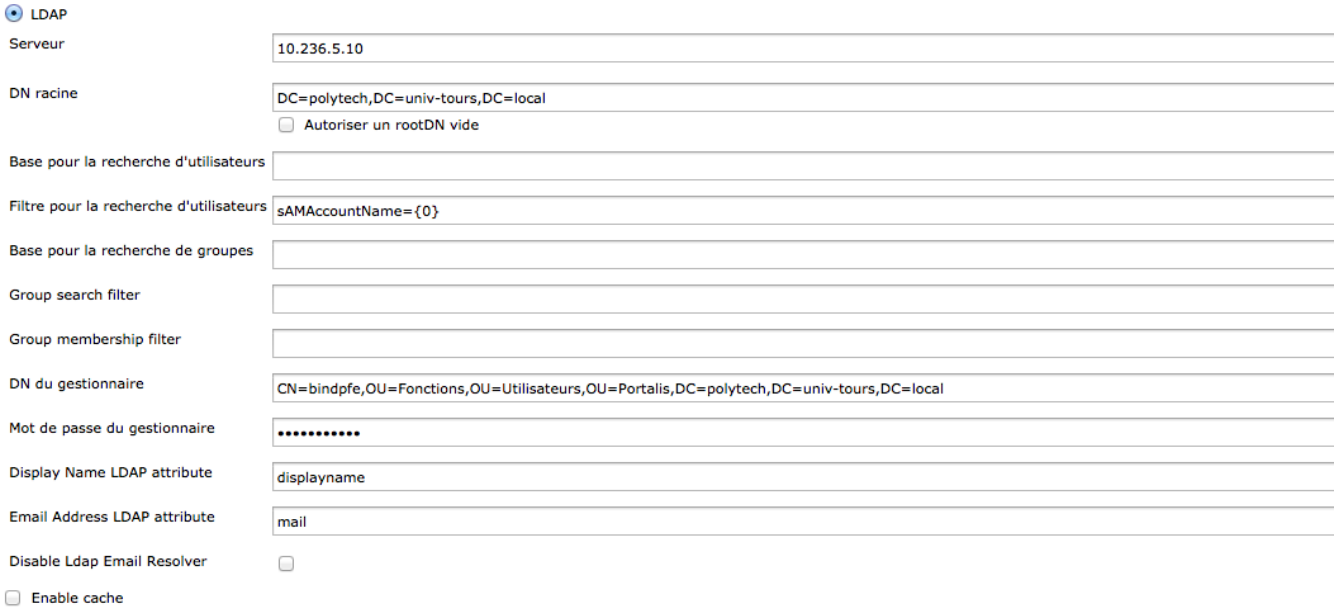

FIGURE 3.8 – Installation Jenkins

Il faut ensuite donner les droits correspondants à chaque profil d'utilisateurs. Pour cela, cochez l'onglet **Sécurité basée sur une matrice**, puis ajoutez au minimum :

1. Un utilisateur (ici bindpfe) qui fera office d'administrateur et aura tous les droits
2. Des accès anonymes (veillez à ne laissez que le minimum...).
3. L'ensemble des groupes pour lesquels chaque utilisateur aura besoin d'accèder à la plateforme. Par exemple, rajoutez le groupe "Groupe_Etudiants_DI4" va permettre à tous les étudiants DI4 d'accèder à la plateforme, si la base de données est à jour.

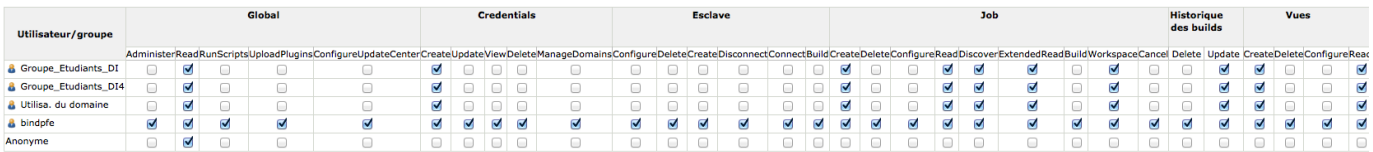

FIGURE 3.9 – Installation Jenkins

NB : Si un élève vient vous voir, et vous indique qu'il n'a pas accès à la plateforme, il suffit qu'il se connecte sur Jenkins avec ses identifiants. Puis qu'il tape l'URL *http ://adresseserveur :8080/jenkins/whoAmI/*. Dedans, il aura le nom exact du groupe dans lequel il se situe, groupe que vous pourrez rajoutez de la même manière que précédemment (comme dans le screenshoot suivant ou il suffira de rajouter "Groupe_Etudiants_DI4".

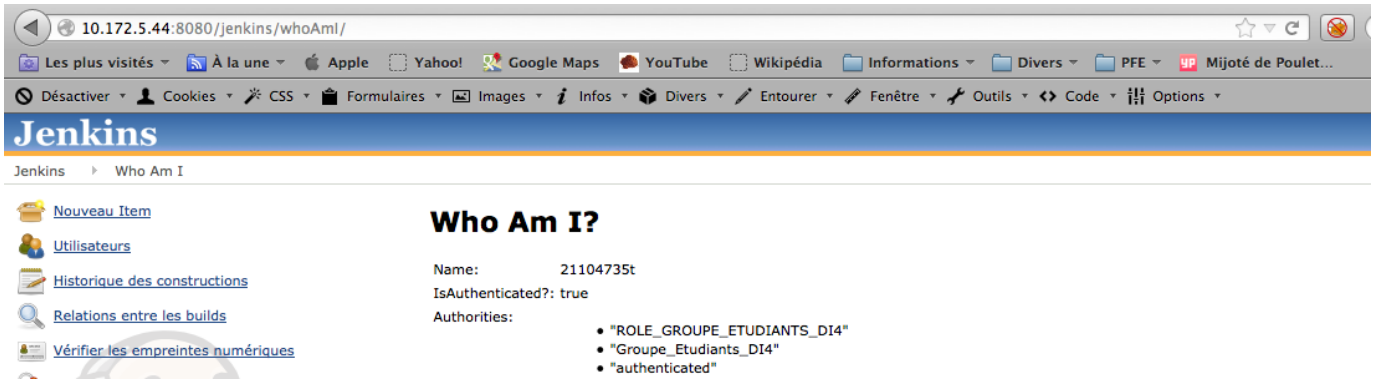

FIGURE 3.10 – Installation Jenkins

Installation SonarQube

Dans cette partie nous allons détailler l'installation de notre outil de qualité de code : Cette installation s'est faite (actuellement) en tant qu'application déployée dans notre conteneur Tomcat7, comme décidé lors de la veille technologique. Cependant une récente décision des développeurs de SonarQube implique que les prochaines versions 4.0 (et plus) ne fonctionneront plus sous Tomcat.Bien que ce ne soit pas gênant pour les versions plus anciennes, celles-ci restent complètes et fonctionnelles sous Tomcat, nous avons détaillé l'installation de SonarQube 4.0 en tant que service.

4.1 Installation PostGreSQL

SonarQube requiert, pour des raisons de performance, l'utilisation d'une base de données comme MySQL, PostGreSQL... La plus utilisée dans ce domaine est PostGreSQL, que l'on a ainsi choisi pour ce projet.
On se rend en premier lieu sur *http ://www.enterprisedb.com/products/pgdownload.do*, sur lequel on récupère la dernière version de PostGreSQL destinée au système d'exploitation Windows x86 (32b).

FIGURE 4.1 – Télécharger PostgreSQL

Une fois téléchargé, on double clique sur l'installateur afin de l'éxécuter. Lorsqu'il sera demandé, **il**

faudra renseigner les login/mots de passe : sonar/sonar. *Note : Il n'est pas nécéssaire d'installer toutes les extensions liées à PostGreSQL.*

Une fois ceci fait, il faut aller dans le paramètrage des variables d'environnement et rajouter une variable **POSTGRES_HOME** ayant pour valeur C :/Program Files/PostgreSQL/9.2/bin

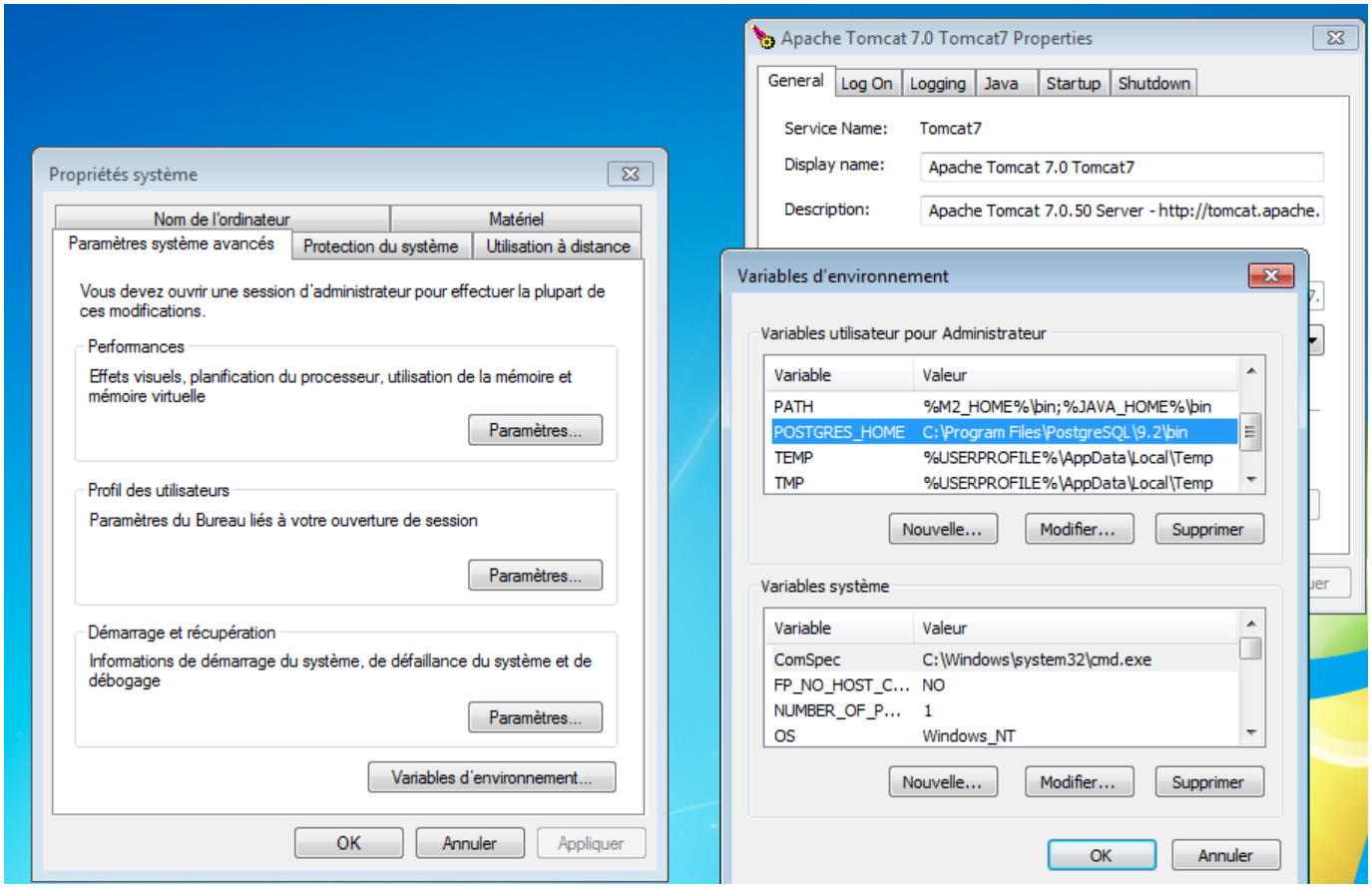

FIGURE 4.2 – Configuration PostgreSQL

Il faut de plus permettre à l'utilisateur postgres d'aller écrire dans son dossier. Pour cela : Click droit sur *C :/Program Files/PostgreSQL*, onglet propriétés, et changer les attributs de sécurité.

.

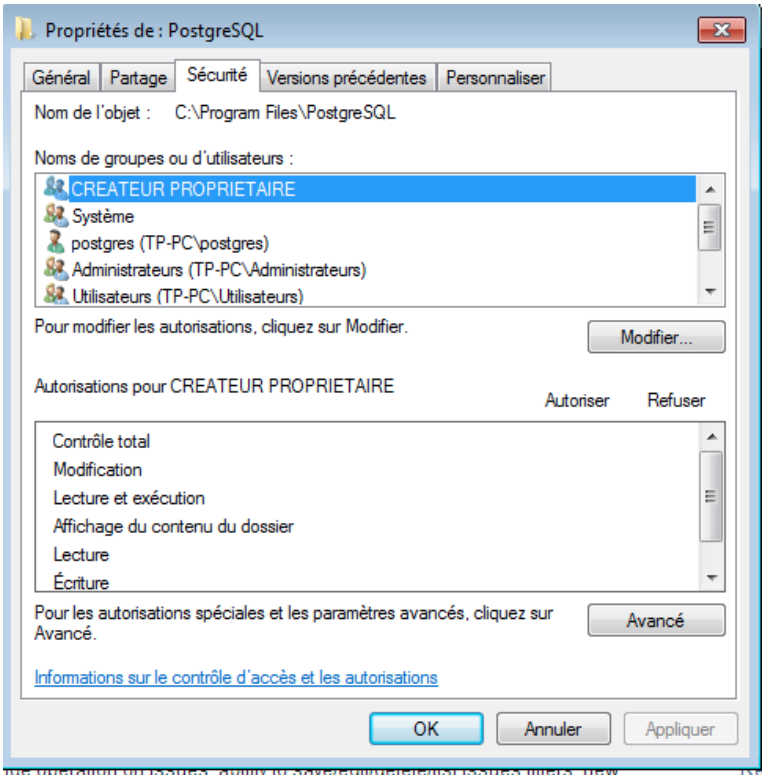

FIGURE 4.3 – Création de base de données nécéssaire à Sonar

Une fois ceci fait, dans un terminal, écrire : *runas /user :postgres cmd*. Le mot de passe demandé est **sonar** (cf installation précédente).

```
c:\Program Files\PostgreSQL\9.2\bin>runas /user:postgres cmd
Entrez le mot de passe de postgres :
Tentative de lancement de cmd en tant qu'utilisateur "TP-PC\postgres" ...
```

FIGURE 4.4 – Création de base de données nécéssaire à Sonar

Dans le nouveau terminal, exécuter cd *"c :/Program Files/PostgreSQL/9.2/bin/"* afin de vous placer dans le bon repertoire ou le reste de l'installation va s'éxécuter. Ensuite, on démarre le contrôleur de postgres grâce à la commande *pg_ctl.exe start -D ../data*. Vous devriez voir le message suivant s'afficher :

```
Succès. Vous pouvez maintenant lancer le serveur de bases de données par :

    "postgres" -D "c:/Program Files/PostgreSQL/9.2/sonar"
ou
    "pg_ctl" -D "c:/Program Files/PostgreSQL/9.2/sonar" -l journal_applicatif
```

FIGURE 4.5 – Création de base de données nécéssaire à Sonar

Si un problème se pose, notamment le fait d'avoir postgres déjà actif sur le même port, il va falloir tuer les processus postgres déjà actifs à l'aide de la commande dans un terminal : *taskkill /PID numeropid /F* (on récupère les numeropid via la commande tasklist).

.

```
explorer.exe                  1436 Console                    1     59 140 Ko
dwm.exe                       1444 Console                    1      6 364 Ko
svchost.exe                   1504 Services                   0     10 140 Ko
taskhost.exe                  1516 Console                    1      5 672 Ko
VMwareTray.exe                1672 Console                    1      4 636 Ko
vmtoolsd.exe                  1680 Console                    1     14 240 Ko
msseces.exe                   1688 Console                    1     10 760 Ko
jusched.exe                   1728 Console                    1      9 556 Ko
Tomcat7w.exe                  1840 Console                    1      3 540 Ko
httpd.exe                     1892 Services                   0     10 352 Ko
svchost.exe                   1948 Services                   0      6 588 Ko
httpd.exe                      392 Services                   0     10 376 Ko
vmtoolsd.exe                  2384 Services                   0     11 652 Ko
postgres.exe                  2528 Services                   0      9 536 Ko
conhost.exe                   2536 Services                   0      2 436 Ko
postgres.exe                  2792 Services                   0      4 324 Ko
postgres.exe                  3108 Services                   0      4 924 Ko
postgres.exe                  3116 Services                   0      5 132 Ko
postgres.exe                  3124 Services                   0      4 888 Ko
postgres.exe                  3132 Services                   0      6 408 Ko
postgres.exe                  3140 Services                   0      4 412 Ko
SearchIndexer.exe             3260 Services                   0     16 320 Ko
TPAutoConnSvc.exe             3388 Services                   0      5 176 Ko
NisSrv.exe                    3448 Services                   0      4 284 Ko
TPAutoConnect.exe             3724 Console                    1      6 892 Ko
conhost.exe                   3732 Console                    1      2 520 Ko
dllhost.exe                   4044 Services                   0      8 428 Ko
msdtc.exe                     2072 Services                   0      6 028 Ko
cmd.exe                       2964 Console                    1      2 300 Ko
conhost.exe                   2956 Console                    1      4 604 Ko
notepad.exe                   3032 Console                    1      5 448 Ko
svchost.exe                   2008 Services                   0      5 760 Ko
sppsvc.exe                    2880 Services                   0      4 568 Ko
audiodg.exe                    460 Services                   0     13 508 Ko
_uninstall3316                3156 Console                    1     59 636 Ko
WmiPrvSE.exe                  2272 Services                   0      5 240 Ko
SearchProtocolHost.exe        2296 Services                   0      6 016 Ko
SearchFilterHost.exe          1232 Services                   0      3 724 Ko
tasklist.exe                  2460 Console                    1      4 160 Ko

c:\Program Files\PostgreSQL\9.2\bin>taskkill /3140
Erreur : Argument ou option non valide - « /3140 ».
Entrez "TASKKILL /?" pour afficher la syntaxe.

c:\Program Files\PostgreSQL\9.2\bin>taskkill /PID 3140 /F
Opération réussie : le processus avec PID 3140 a été terminé.

c:\Program Files\PostgreSQL\9.2\bin>taskkill /PID 3132 /F
Opération réussie : le processus avec PID 3132 a été terminé.
```

FIGURE 4.6 – Création de base de données nécéssaire à Sonar

On peut vérifier que postgres est bien en train d'écouter sur le port par défault (5432) netstat -aon | findstr 127.0.0.1 :5432

```
c:\Program Files\PostgreSQL\9.2\bin>netstat -aon | findstr 127.0.0.1:5432
  TCP    127.0.0.1:5432         0.0.0.0:0              LISTENING       1076
```

FIGURE 4.7 – Création de base de données nécéssaire à Sonar

Ensuite on se connecte à l'éditeur psql en tapant la commande psql. On crée notre utilisateur sonar et auquel on alloue tous les droits.

.

On éxécute ensuite les commandes suivantes :

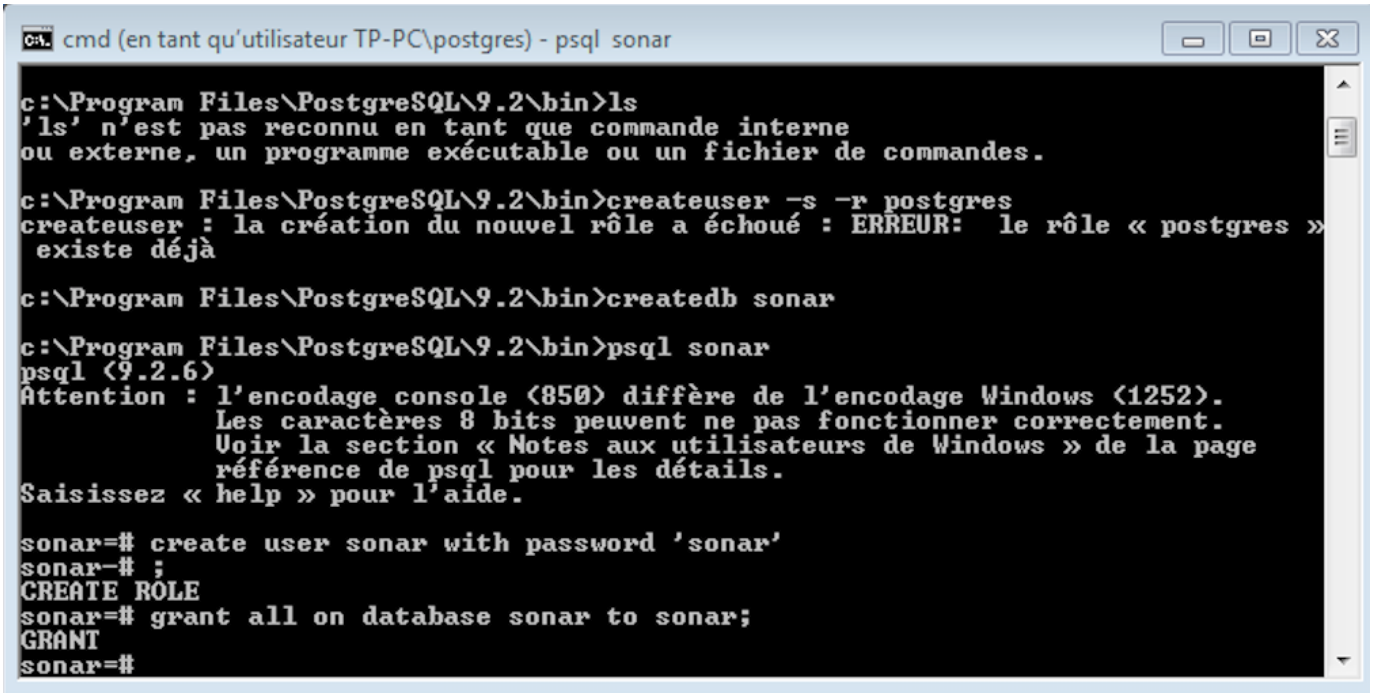

FIGURE 4.8 – Création de base de données nécéssaire à Sonar

CREATE USER sonar with password 'sonar' ;
CREATE DATABASE sonar WITH OWNER sonar ENCODING 'UTF8' ;
On vérifie que cela s'est bien effectué :
select usename,usesysid FROM PG_USER ;
select datname, datdba, encoding FROM pg_database ;
On obtient ainsi :

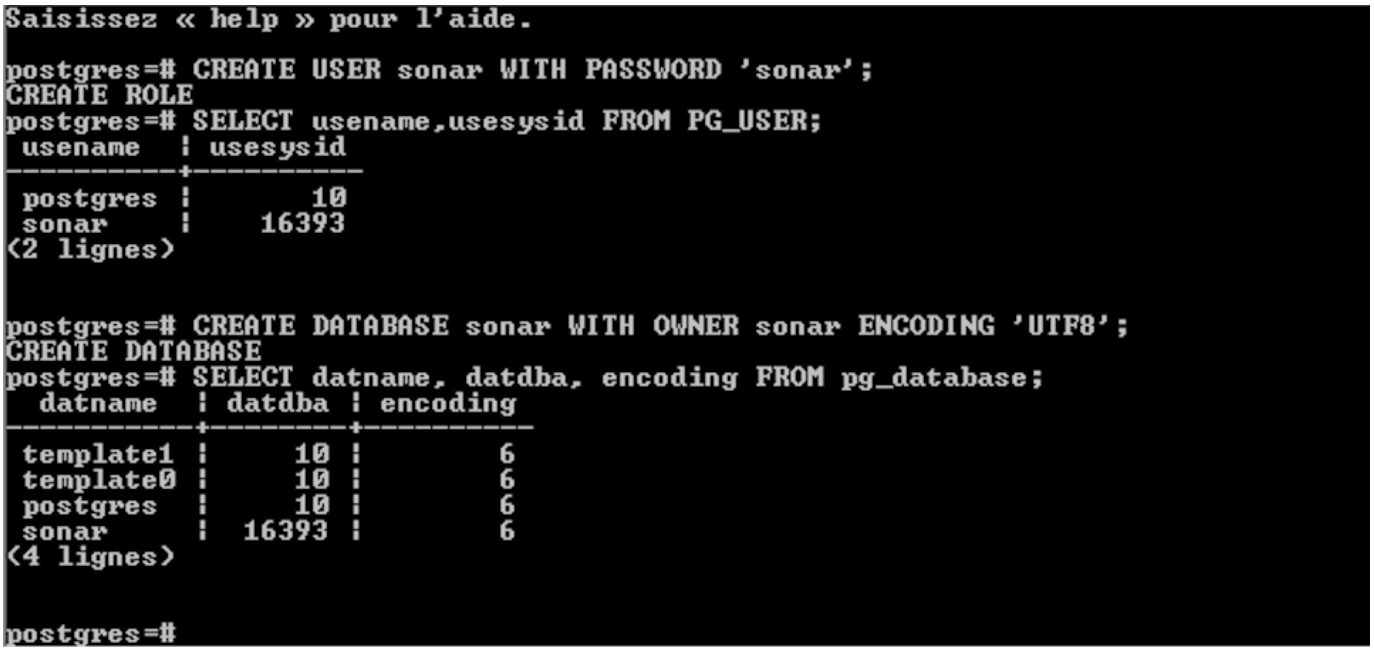

FIGURE 4.9 – Création de base de données nécéssaire à Sonar

On quitte psql à l'aide de la commande \q. Il ne faut d'ailleurs pas oublier que pour la suite de cette

installation, PostGreSQL doit être lancé ! Si ce n'est pas le cas, il faut lancer PostGreSQL depuis les services de windows, comme le montre le screenshot suivant.

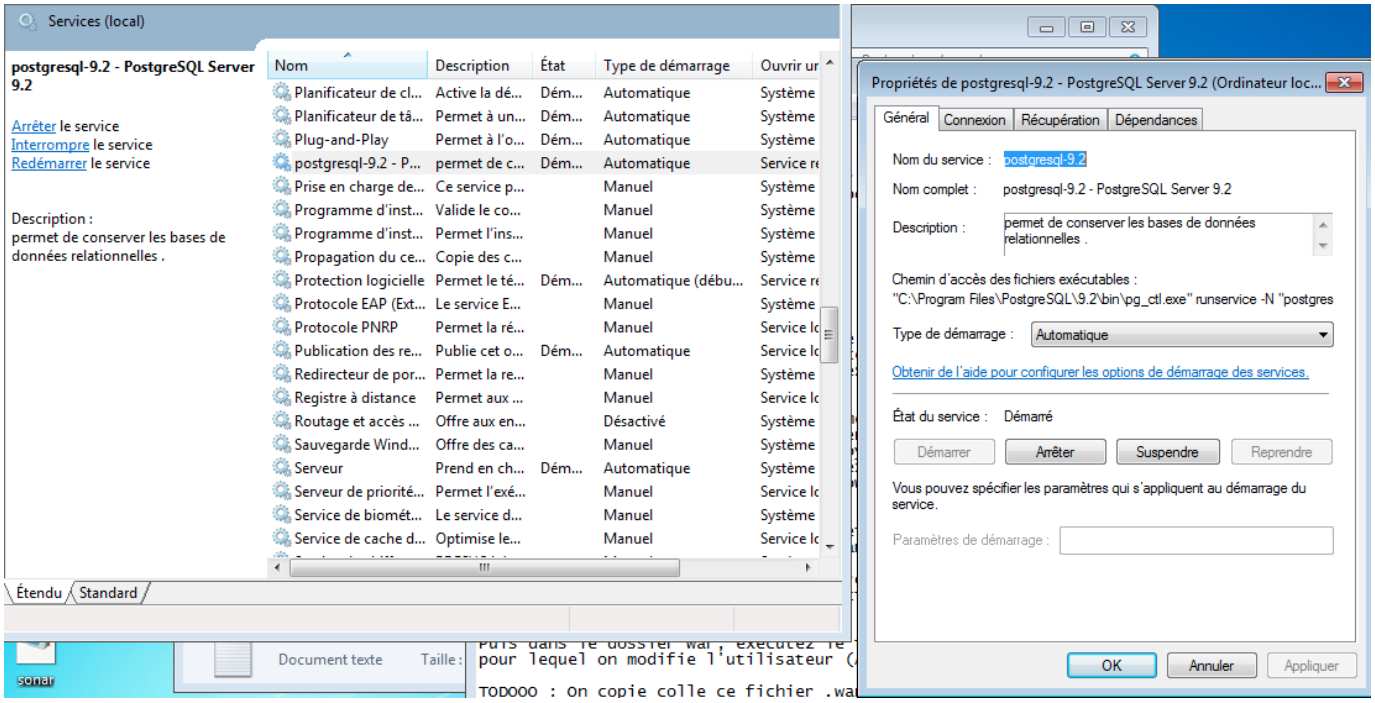

FIGURE 4.10 – Lancer PostgreSQL

4.2 Déploiement de SonarQube dans Tomcat

Il est tout à fait possible d'installer SonarQube en tant que service Windows. Cela sera d'ailleurs même recommandé à l'avenir (comme vu dans l'introduction de cette partie). Cependant, rien n'empêche à Tomcat d'éxécuter SonarQube (version 3.x.x), et nous allons donc l'installer de cette manière. De plus, la migration en tant que Service Windows est excessivement facile, et pourra facilement être abordée par l'équipe informatique de Polytech. Elle est d'ailleurs détaillée dans la partie suivante.

Rendez vous sur *http ://www.sonarqube.org/downloads/* (et téléchargez une version antèrieure à 4.0).

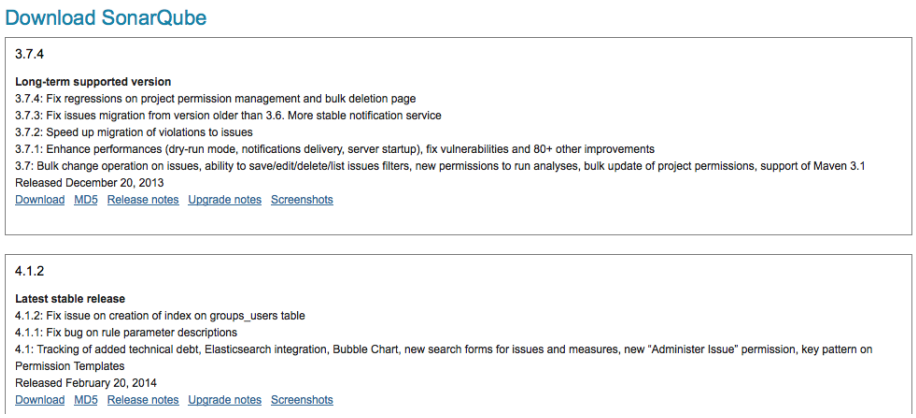

FIGURE 4.11 – Téléchargez sonar

Décompressez ce dossier vers *C :/Program Files/SonarQube*. On modifie ensuite le fichier **sonar.properties** (qui se trouve dans le repertoire conf). Il nous suffit juste de commenter l'activation par défaut et décommenter l'accès à la base de données PostgreSQL. On y règle aussi la question de l'authentification en rajoutant les lignes suivantes :

```
#--------------------------------------------------------
# LDAP configuration
#--------------------------------------------------------

# General Configuration
sonar.security.realm=LDAP
ldap.url=ldap://10.236.5.10:389
ldap.bindDn=CN=bindpfe,OU=Fonctions,OU=Utilisateurs,OU=Portalis,DC=polytech,DC=univ-tours,DC=local
ldap.bindPassword=bindpfe2014


# User Configuration
ldap.user.baseDn=DC=polytech,DC=univ-tours,DC=local
ldap.user.request=(sAMAccountName={0})

# Group Configuration
ldap.group.baseDn=DC=polytech,DC=univ-tours,DC=local
```

FIGURE 4.12 – Téléchargez sonar

Puis dans le dossier war, exécutez le fichier build-war. Celui-ci produit un fichier sonar.war, pour lequel on modifie l'utilisateur et droits de propriétés (à l'instar de postgreSQL).
On copie colle ce fichier .war dans le dossier webapps de tomcat7.

On peut à présent relancer tomcat. Si tout se passe bien, on peut accèder à l'interface web de sonar et jenkins via un navigateur :
http ://localhost :8080/sonar/, *http ://localhost :8080/jenkins/*.

On doit ensuite configurer le plugin sonar de Jenkins, pour cela réferez vous à la partie Jenkins de ce manuel (partie plugins).

4.3 Installation de SonarQube en tant que service

Pour lancer SonarQube en tant que service c'est encore plus simple, http ://www.sonarqube.org/downloads/ (et téléchargez une version supèrieur à 4.0).
Décompressez ce dossier vers C :/Program Files/SonarQube.
On modifie ensuite le fichier sonar.properties (qui se trouve dans le repertoire conf). Il nous suffit juste de commenter l'activation par défaut et décommenter l'accès à la base de données PostgreSQL

Attention : Il faut également recommencer l'étape de création de la base de données sonar. Pour cela supprimez la base de données sonar : (commande *dropdb sonar*, n'oubliez pas d'être loggué en utilisateur postgres et d'être dans le dossier bin de postgres..), et éteignez postgres depuis la fenêtre Service de Windows. On reprend ensuite depuis l'étape *pg_ctl.exe start -D ../data*..
Pour plus d'informations réfèrez vous à la partie portant sur PostGreSQL.

Ensuite, vous pouvez relancer sonar. Pour cela il suffit d'éxécutez l'application StartSonar (après avoir utilisé les installateurs) situés dans le dossier bin, et d'accèder à l'adresse *http ://localhost :9000*.

.

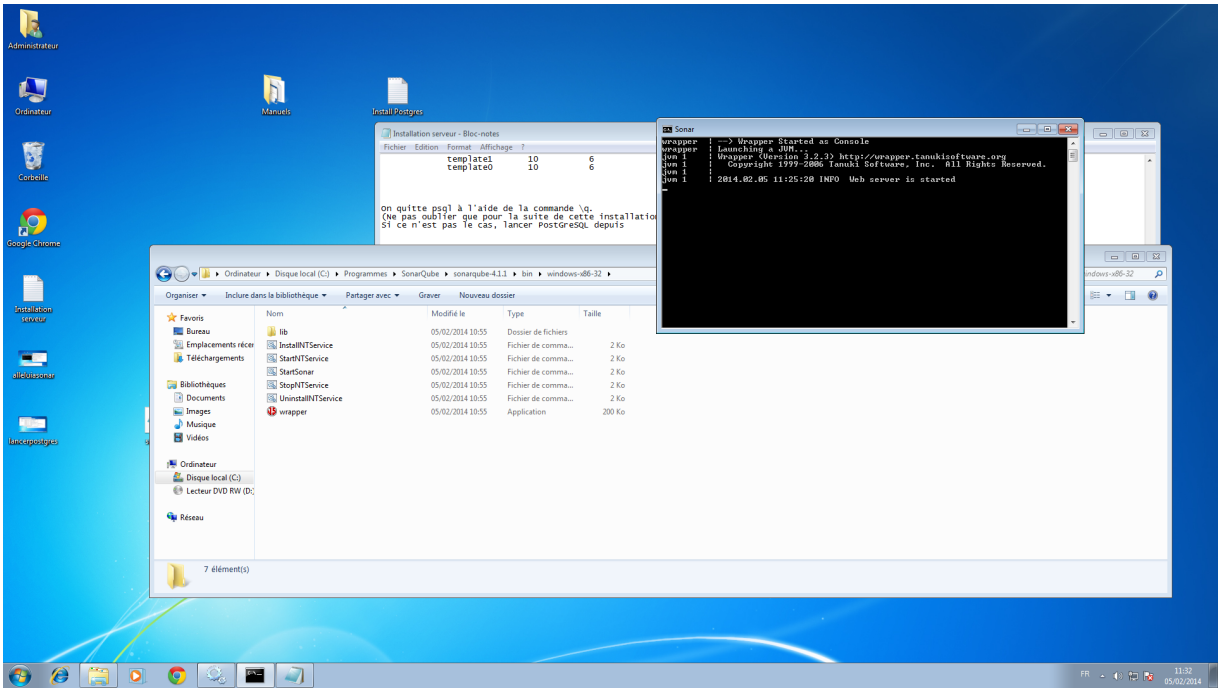

FIGURE 4.13 – Lancer sonar

4.4 Configuration de SonarQube

Une fois sonarqube lancé, on installe les plugins nécéssaires et choisis à la suite de la réunion tenue courant janvier. Pour cela, il faut se connecter en tant qu'admin/admin (par défaut) sur l'interface web de sonarqube. Allez ensuite dans settings, puis Update center (cf screenshots).

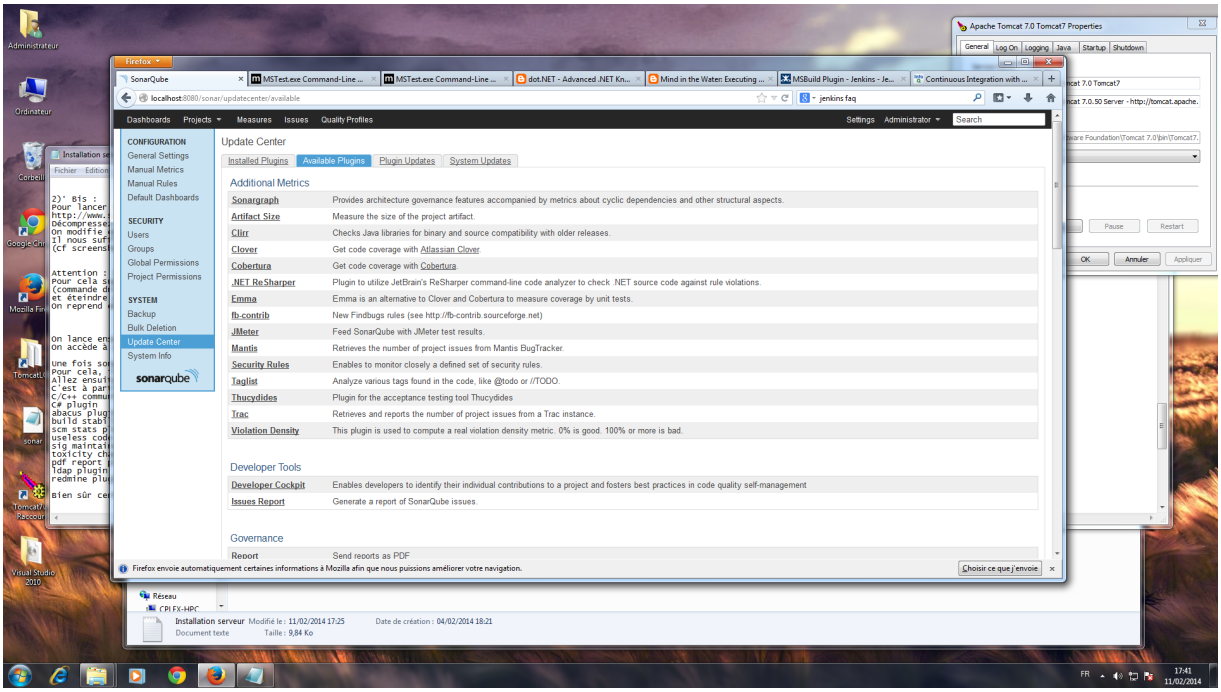

FIGURE 4.14 – Sonar Plugin

C'est à partir de cette interface que l'on installe la liste des plugins suivants :

— C/C++ community
— C# plugin
— abacus plugin
— build stability plugin
— scm stats plugin
— useless code tracker plugin
— sig maintainability model
— toxicity chart plugin
— pdf report plugin
— ldap plugin
— redmine plugin

Autres

5.1 Installation de Visual Studio

Pour le bon fonctionnement sous Jenkins des projets construits sous visual studio, il est nécéssaire d'installer Visual Studio 2010 (se référer à l'équipe du service informatique pour récupèrer le logiciel), ainsi que MSbuild v4.0.

Avant toute chose, désinstallez tout programme C++ (Panneau de Configuration > Programmes et fonctionnalités et click droit sur les applications à supprimer).

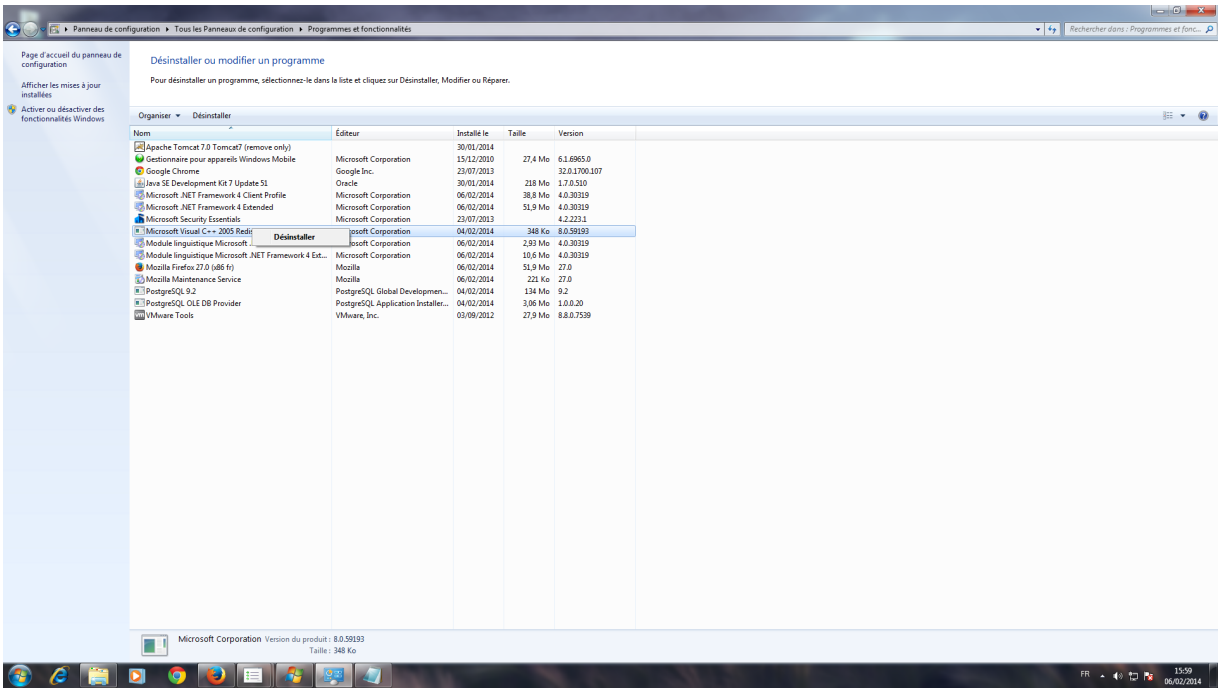

FIGURE 5.1 – Desinstallation VC++

Lors de l'installation du SDK décochez toutes fonctionnalités relatives à C++.
http ://www.microsoft.com/fr-fr/download/details.aspx ?id=17851
http ://www.microsoft.com/en-us/download/details.aspx ?id=8279

Ensuite on peut réinstaller les outils C++ désinstallés auparavant (et qui seront nécéssaires pour la suite).

5.2 Connexion

Pour être capable d'accèder au serveur situé sur une VM depuis une machine cliente, on doit se situer dans le même réseau que le serveur **(pour le moment)**. On doit donc passer la connection de la machine vituelle en mode Bridged, ce qui se fait dans les paramètres de la machine virtuelle (settings de VMWare Fusion). Cela permettra à la machine virtuelle (notre serveur) de partager l'adresse ip de son hôte physique et donc de le contacter (si l'on est dans le même réseau).

Bien sûr avec le déménagement prévu de notre machine pour que celle-ci soit accessible depuis l'extérieure, cette partie sera amenée à être modifiée.

Conclusion

L'installation de l'ensemble de nos outils sur notre serveur est ainsi faite, permettant à n'importe quel client d'accèder aux outils d'intégration continue et qualité de code, pour peu que ce client connaisse l'adresse réseau du serveur et soit capable de communiquer avec ce dernier. Le lancement de l'ensemble des applications se fait ainsi via l'éxécutable fourni par Tomcat, *tomcatw*, permettant de démarrer Tomcat.

Serveur Intégration Continue et Qualité de Code

Département Informatique
5e année
2008 - 2009

Manuel Installation

Résumé : Manuel d'installation du Serveur d'Intégration Continue et Qualité de Code

Mots clefs : Manuel d'installation du serveur

Abstract: Installation Manual (Server)

Keywords: Installation Manuel (Server)

Encadrants
Nicolas Ragot
nicolas.monmarche@univ-tours.fr
Vincent T'Kindt
tkindt@univ-tours.fr

Étudiants
Anne CASTIER
anne.castier@etu.univ-tours.fr

DI5 2013 - 2014

Université François-Rabelais, Tours

École Polytechnique de l'Université de Tours
64, Avenue Jean Portalis
37200 TOURS, FRANCE
Tél. +33 (0)2 47 36 14 14
www.polytech.univ-tours.fr

Département Informatique
5e année
2013 - 2014

Manuel d'installation de la Machine Virtuelle Cliente

Serveur d'Intégration Continue et Qualité de Code, côté client

Encadrants
Nicolas Ragot
nicolas.monmarche@univ-tours.fr
Vincent T'Kindt
tkindt@univ-tours.fr

Étudiants
Anne CASTIER
anne.castier@etu.univ-tours.fr

DI5 2013 - 2014

Université François-Rabelais, Tours

Version du 16 avril 2014

Table des matières

Introduction

Ce manuel relate l'installation de départ de la partie cliente, à même d'utiliser le serveur d'intégration continue et de qualité de code (installé préalablement). Cette installation se fait sur une machine virtuelle fonctionnant sous Windows 7. **Le choix de ce système d'exploitation est lié à la volonté de pouvoir travailler avec des environnements microsoft comme Visual Studio et le langage C#.** De plus, vous verrez que cette installation est très rapide, elle met surtout en place des installateurs et n'a que peu de fichiers de configuration à modifier.
NB : On peut retrouver facilement une machine virtuelle de base depuis les machines virtuelles sources proposées par l'école, sur laquelle on pourra démarrer l'installation telle que décrite par ce manuel.

Pré-installations nécessaires

Afin de pouvoir utiliser le serveur, certaines pré-installations sont nécessaires.

2.1 Installation java 1.7

Voici le pas à pas de l'installation du jdk 1.7.
Rendez vous d'abord sur la page suivante : *http ://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html*

Java SE Development Kit 7u51

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

Thank you for accepting the Oracle Binary Code License Agreement for Java SE; you may now download this software.

Product / File Description	File Size	Download
Linux ARM v6/v7 Hard Float ABI	67.7 MB	jdk-7u51-linux-arm-vfp-hflt.tar.gz
Linux ARM v6/v7 Soft Float ABI	67.68 MB	jdk-7u51-linux-arm-vfp-sflt.tar.gz
Linux x86	115.65 MB	jdk-7u51-linux-i586.rpm
Linux x86	132.98 MB	jdk-7u51-linux-i586.tar.gz
Linux x64	116.96 MB	jdk-7u51-linux-x64.rpm
Linux x64	131.8 MB	jdk-7u51-linux-x64.tar.gz
Mac OS X x64	179.49 MB	jdk-7u51-macosx-x64.dmg
Solaris x86 (SVR4 package)	140.02 MB	jdk-7u51-solaris-i586.tar.Z
Solaris x86	95.13 MB	jdk-7u51-solaris-i586.tar.gz
Solaris x64 (SVR4 package)	24.53 MB	jdk-7u51-solaris-x64.tar.Z
Solaris x64	16.28 MB	jdk-7u51-solaris-x64.tar.gz
Solaris SPARC (SVR4 package)	139.39 MB	jdk-7u51-solaris-sparc.tar.Z
Solaris SPARC	98.19 MB	jdk-7u51-solaris-sparc.tar.gz
Solaris SPARC 64-bit (SVR4 package)	23.94 MB	jdk-7u51-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	18.33 MB	jdk-7u51-solaris-sparcv9.tar.gz
Windows x86	123.64 MB	jdk-7u51-windows-i586.exe
Windows x64	125.46 MB	jdk-7u51-windows-x64.exe

FIGURE 2.1 – JDK - Java

Acceptez les conditions et installez la version **Windows x86.**
Exécutez ensuite l'éxécutable, suivez l'installation. Laissez le repertoire d'installation par défaut (à savoir C :/Program Files/Java/jdk.1.7.0_51). .

Ajoutez une **variable d'environnement utilisateur** JAVA_HOME pointant sur C :/Program Files/Java/jdk.1.7.0_51

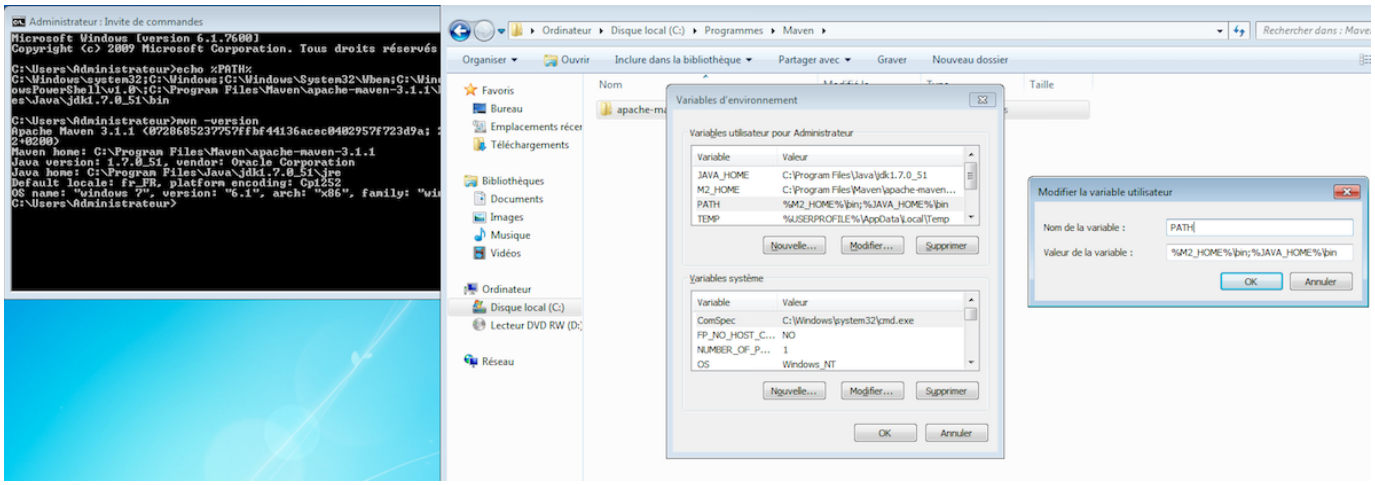

FIGURE 2.2 – Variables JDK - Java

De plus, il est necéssaire d'aller dans le **panneau de configuration windows**, onglet **Java**. Décochez le jre actuel, et ajoutez celui installé : Cliquez sur l'onglet recherche et allez chercher le dossier jdk installé.

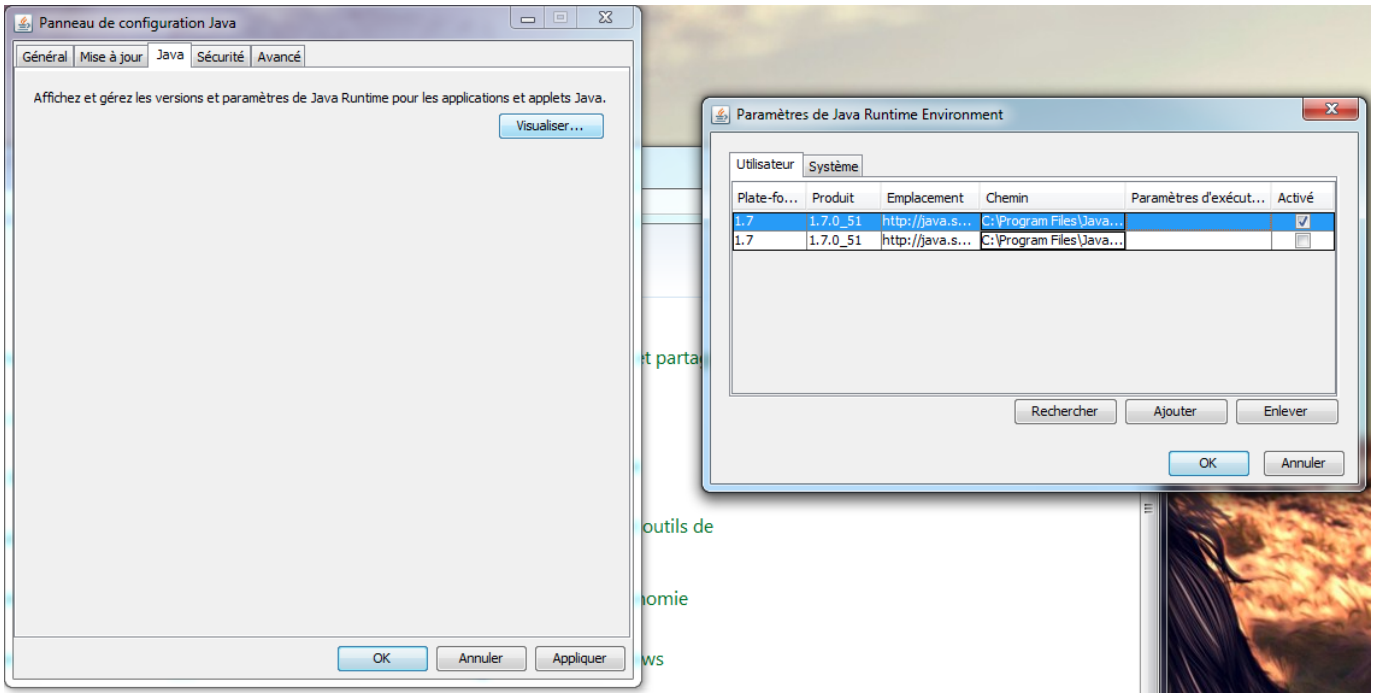

FIGURE 2.3 – JRE Configuration - Java

.

Voilà, l'installation du JDK est finie ! Si tout s'est bien passé, lancez une invite de commandes (un terminal...), et éxécutez la commande *java -version*. Le résultat affiché devrait être notre version de Java (la 1.7.51). On peut à présent passer à celle de Maven, autre incontournable pour notre serveur.

2.2 Installation Maven

L'installation de Maven est très rapide.
Il faut tout d'abord télécharger une version récente de Maven à l'adresse suivante :
http ://maven.apache.org/download.cgi (**Choisissez bien la version bin.**)

Maven 3.2.1

This is the current stable version of Maven.

	Link	Checksum	Signature
Maven 3.2.1 (Binary tar.gz)	apache-maven-3.2.1-bin.tar.gz	apache-maven-3.2.1-bin.tar.gz.md5	apache-maven-3.2.1-bin.tar.gz.asc
Maven 3.2.1 (Binary zip)	apache-maven-3.2.1-bin.zip	apache-maven-3.2.1-bin.zip.md5	apache-maven-3.2.1-bin.zip.asc
Maven 3.2.1 (Source tar.gz)	apache-maven-3.2.1-src.tar.gz	apache-maven-3.2.1-src.tar.gz.md5	apache-maven-3.2.1-src.tar.gz.asc
Maven 3.2.1 (Source zip)	apache-maven-3.2.1-src.zip	apache-maven-3.2.1-src.zip.md5	apache-maven-3.2.1-src.zip.asc
Release Notes	3.2.1		
Release Reference Documentation	3.2.1		

FIGURE 2.4 – Maven Installation

Dézippez vers **C :/Program Files/Java/Program Files/Maven** (Vous devrez créer le dossier). Ajoutez une variable d'environnement utilisateur **M2_HOME** pointant sur C :/Program Files/Maven/apache-maven-3.1.1 (cf screenshots). Ajoutez aussi une variable d'environnement utilisateur **PATH** contenant %M2_HOME%/bin ;%JAVA_HOME%/bin

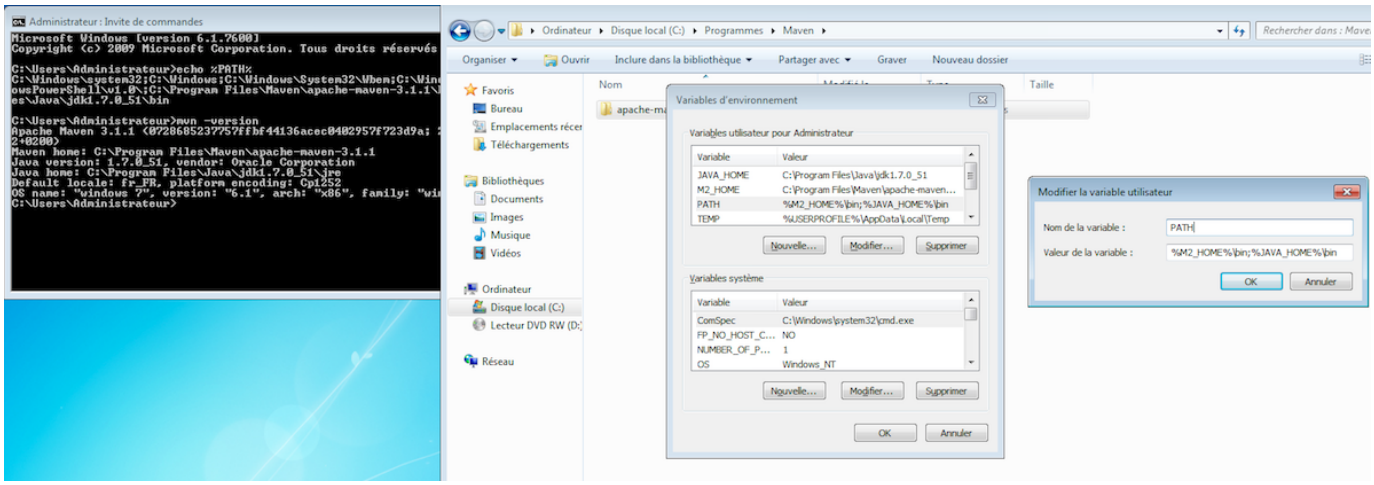

FIGURE 2.5 – Variables JDK - Java

Vérifiez la bonne installation à l'aide de la commande *mvn –version* dans un terminal (invite de commandes). Vient ensuite l'étape de configuration : Configurez le fichier settings.xml (dans le dossier conf) du dossier Maven.
Remplacez par :
<settings xmlns="http ://maven.apache.org/SETTINGS/1.0.0"
xmlns :xsi="http ://www.w3.org/2001/XMLSchema-instance"
xsi :schemaLocation="http ://maven.apache.org/SETTINGS/1.0.0
http ://maven.apache.org/xsd/settings-1.0.0.xsd">
<localRepository/>

<interactiveMode/>
<usePluginRegistry/>
<offline/>
<pluginGroups/>
<servers/>
<mirrors/>
<proxies/>
<profiles/>
<activeProfiles/>
</settings>

2.3 Installation TortoiseSVN

Afin de pouvoir utiliser le serveur subversion fourni avec le Redmine de l'école, il peut être utile (mais pas nécessaire) d'installer un logiciel client SVN. D'ailleurs, toutes les manipulations du manuel utilisateur seront faites à l'aide de ce client TortoiseSVN.

2.3.1 Installation

L'installation est très simple : Télechargez la dernière version **32-bits** à l'adresse suivante ; *http ://tortoisesvn.net/downloads.html*

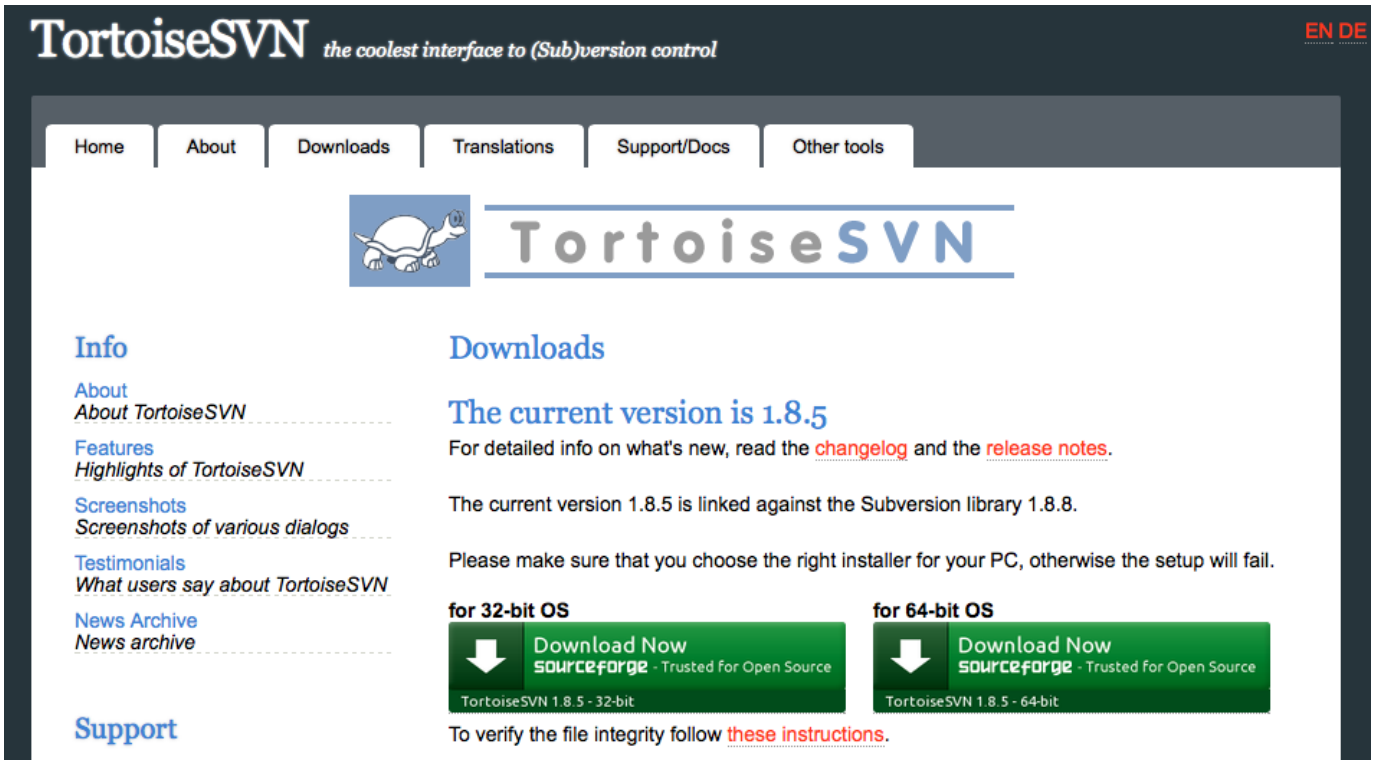

FIGURE 2.6 – Installation SVN Tortoise

Lancez l'éxécutable et suivez toutes les étapes d'installations. Une fois ces étapes finies, SVNTortoise est installé et opérationnel !

Installation Sonar Runner

Téléchargez Sonar Runner depuis :
http ://docs.codehaus.org/display/SONAR/Installing+and+Configuring+SonarQube+Runner (Vous pouvez considèrer que les prérequis (**java/sonarqube**) demandés pour SonarRunner sont déjà présents, puisque l'on va se servir de **SonarQube** installé sur notre serveur !).

Installing and Configuring SonarQube Runner

Créée par David RACODON, dernière modification par Evgeny Mandrikov le janv. 16, 2014

Name	SonarQube Runner
Latest version	**2.3** (23 July 2013)
Requires SonarQube version	**3.0** or higher (check version compatibility)
Download	http://repo1.maven.org/maven2/org/codehaus/sonar/runner/sonar-runner-dist/2.3/sonar-runner-dist-2.3.zip
License	GNU LGPL 3
Developers	Evgeny Mandrikov, Simon Brandhof, Fabrice Bellingard
Issue tracker	http://jira.codehaus.org/browse/SONARPLUGINS/component/14640
Sources	https://github.com/Sonarsource/sonar-runner

FIGURE 3.1 – Microsoft Windows SDK

Dézippez le téléchargement vers le dossier Program Files (situé à la racine C :/). Pour l'occasion, créez lui un dossier dédié tel que *C :/Program Files/SonarRunner* .

Vient ensuite l'étape de configuration :
On va modifier le fichier de configuration de SonarRunner, situé dans le repertoire conf (dans le dossier précédemment dézippé). On y indique l'adresse de notre serveur sonar et la base de données PostgreSQL. Il faut également décommenter l'utilisation d'UTF8 (très important !).

.

```
#Configure here general information about the environment,
such as SonarQube DB details for example
#No information about specific project should appear here

#----- Default SonarQube server
sonar.host.url=http://10.172.5.51:8080/sonar

#----- PostgreSQL
sonar.jdbc.url=jdbc:postgresql://10.172.5.51:5432/sonar
sonar.jdbc.driverClassName=org.postgresql.Driver


#----- MySQL
#sonar.jdbc.url=jdbc:mysql://localhost:3306/sonar?
useUnicode=true&amp;characterEncoding=utf8

#----- Oracle
#sonar.jdbc.url=jdbc:oracle:thin:@localhost/XE

#----- Microsoft SQLServer
#sonar.jdbc.url=jdbc:jtds:sqlserver://localhost/sonar;SelectMe
thod=Cursor

#----- Global database settings
#sonar.jdbc.username=sonar
#sonar.jdbc.password=sonar
```

FIGURE 3.2 – Microsoft Windows SDK

Une fois le fichier sauvegardé, on va ajouter une variable d'environnement SONAR_RUNNER_HOME pointant sur C :/Program Files/SonarRunner/sonar-runner-2.3 (comme on l'a fait pour Maven et notre JDK).

.

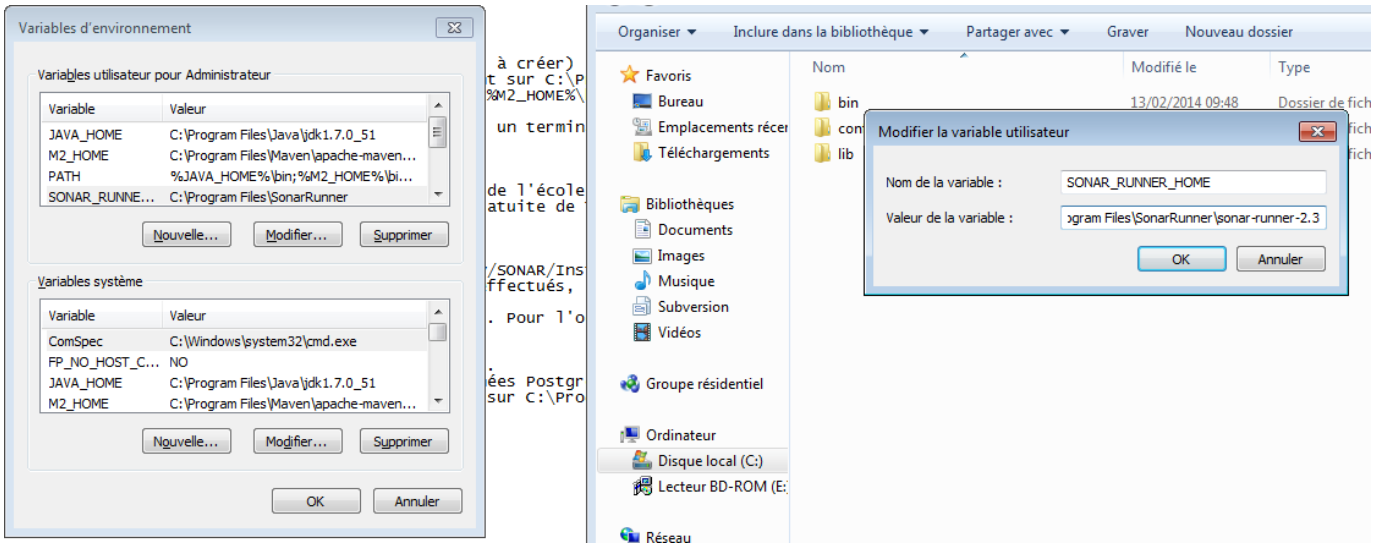

FIGURE 3.3 – Microsoft Windows SDK

.

On modifie ensuite la variable d'environnement PATH en rajoutant une ligne %SONAR_RUNNER_HOME%/bin

.

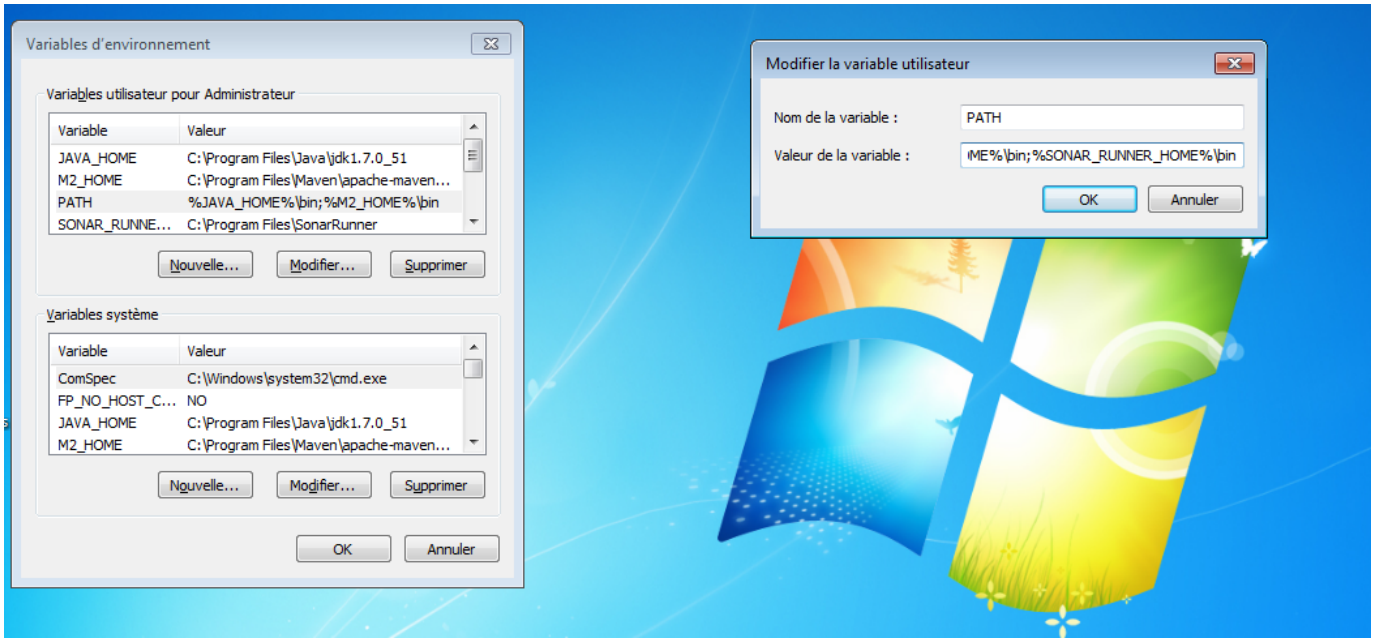

FIGURE 3.4 – Microsoft Windows SDK

.

Testez ensuite le bon fonctionnement de l'installation à l'aide de la commande sonar-runner -h dans un terminal.

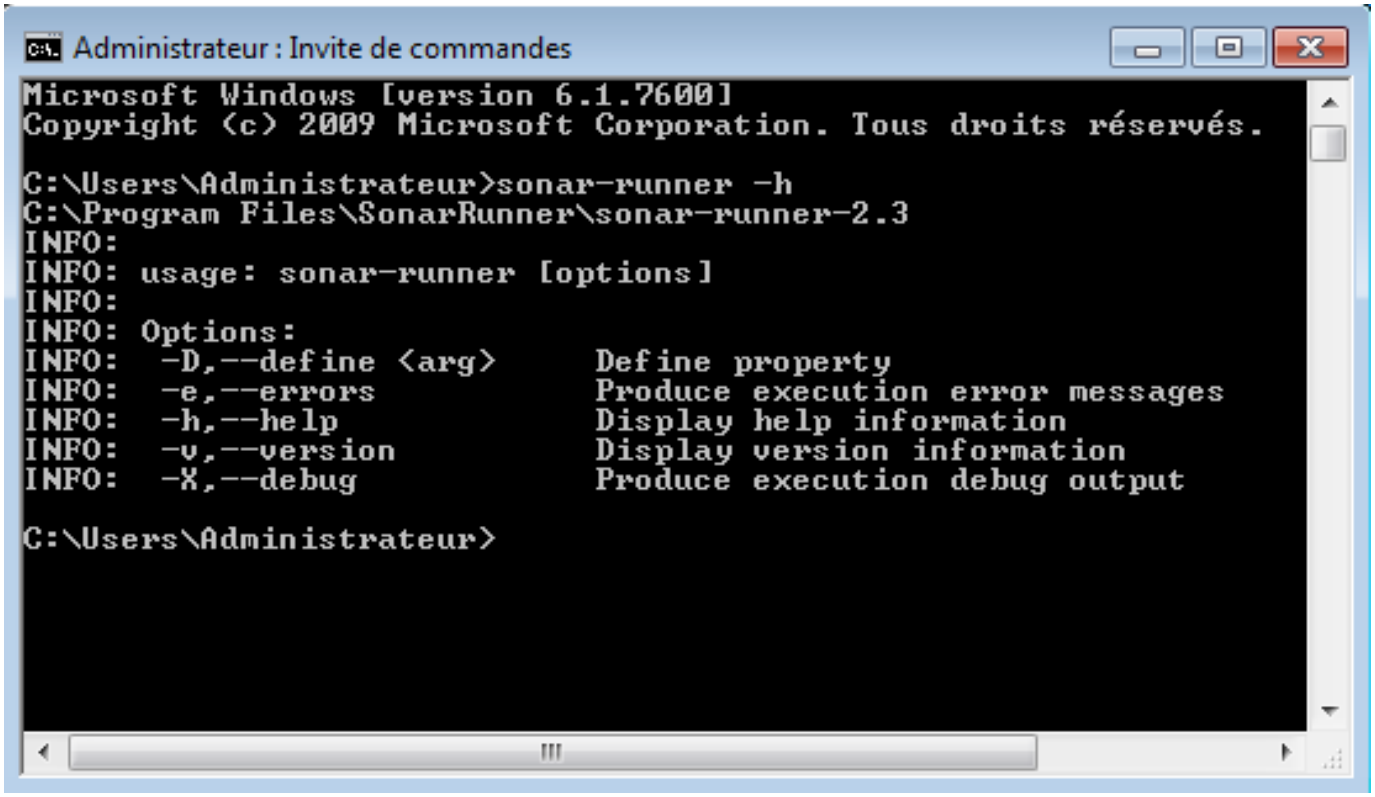

FIGURE 3.5 – Vérification de SonarRunner

Autres

4.1 Installation Eclipse

Cette partie est optionnelle, vous êtes bien sûr tout à fait libre d'installer un autre IDE supportant le langage Java.

Sinon, rendez vous à : *https ://www.eclipse.org/downloads/* et téléchargez la version **Eclipse IDE for Java EE Developers 32-bits.**

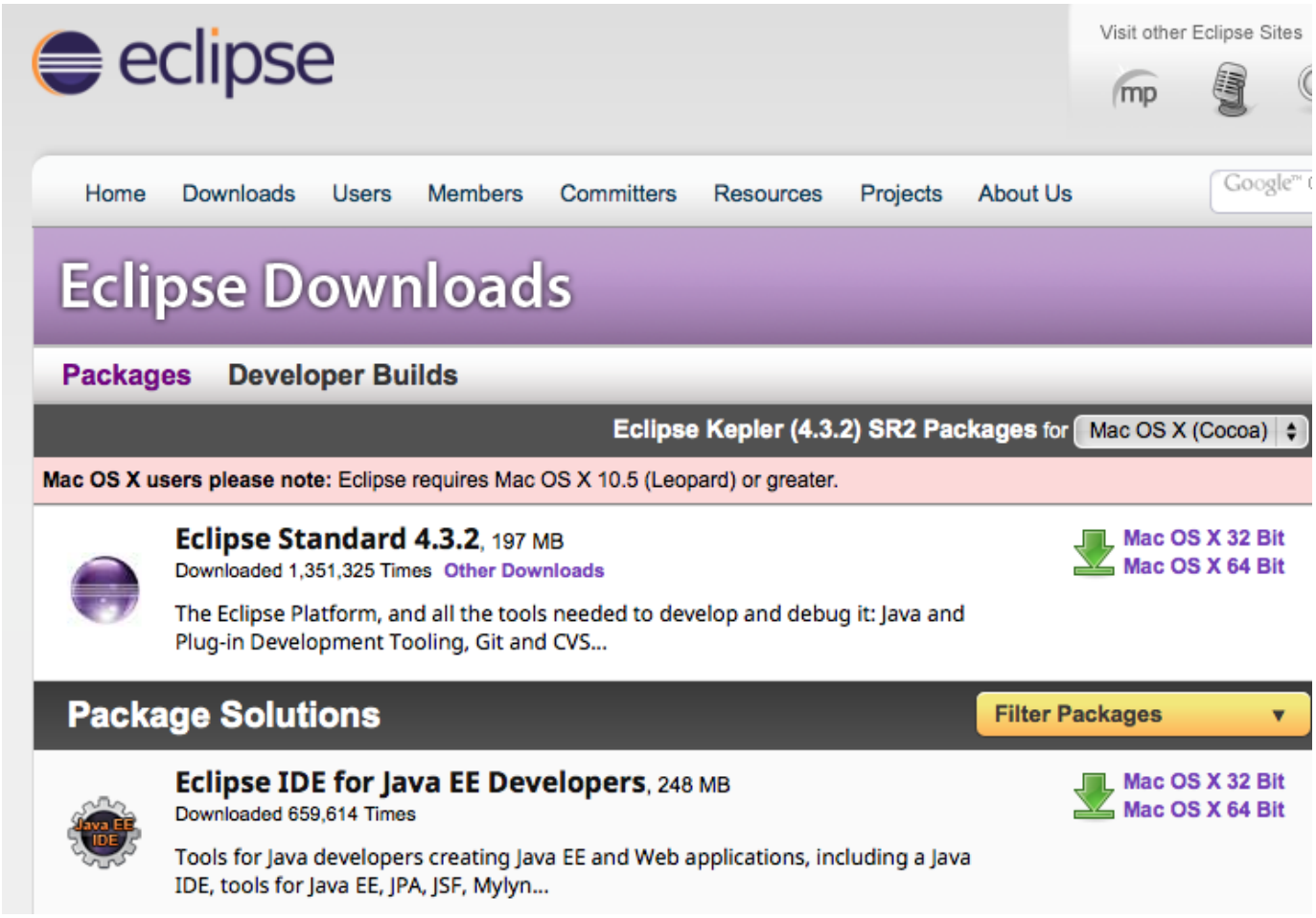

FIGURE 4.1 – Installation SVN Tortoise

Dézipppez le .zip téléchargé vers C :/Program Files (si besoin créez un dossier eclipse !).

4.2 Installation Visual Studio 2010 ou 2012

Préambule : Il faudra choisir une version ou l'autre pour votre machine cliente. En effet l'installation de visual studio 2010 puis 2012 peut amener des défaillances au niveau des tests

lancés pour des projets c/c++ dans visual studio 2010. Selon votre choix, intéressez-vous à la première ou seconde partie.

4.2.1 Installation de Visual Studio 2010

Procurez-vous un .iso de visual studio 2010 auprès du service informatique de l'école (ou votre MSDNAA). Montez cette image, et suivez l'installation.

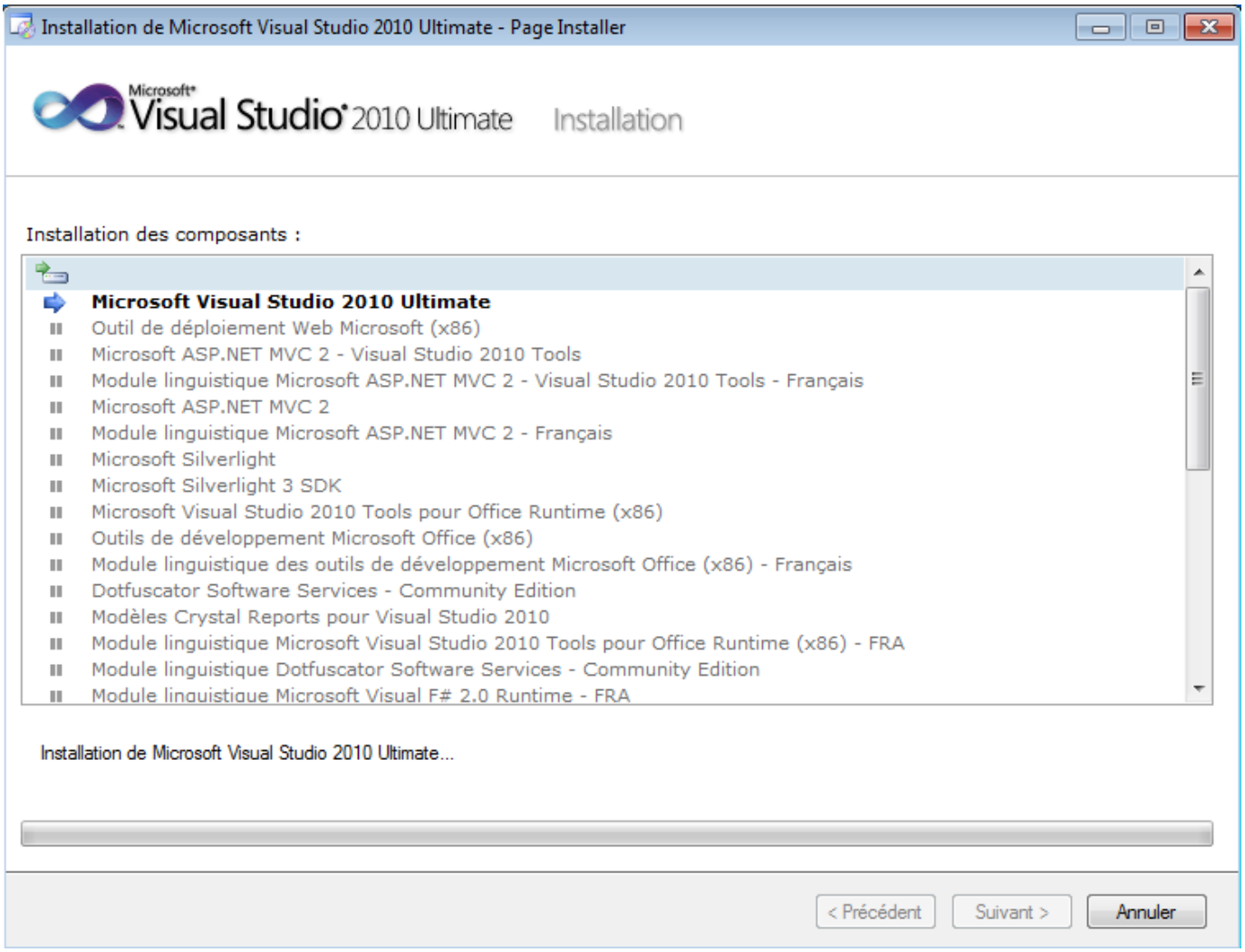

FIGURE 4.2 – Installation SVN Tortoise

4.2.2 Installation de Visual Studio 2012

Procurez-vous un .iso de visual studio 2012 auprès du service informatique de l'école (ou votre MSDNAA). Montez cette image, et suivez l'installation.

4.3 Prérequis pour le langage C#

4.3.1 Avec une machine cliente avec visual studio 2010

Pour le bon fonctionnement de SonarRunner sous C#, il est nécessaire d'installer 4 plugins.

— Gallio : Appliquez l'éxécutable que vous trouverez à : *https ://code.google.com/p/mb-unit/downloads/list*
— StyleCop : Appliquez l'éxécutable que vous trouverez à : *http ://stylecop.codeplex.com/*
— OpenCover : Appliquez l'éxécutable que vous trouverez à : *http ://opencover.codeplex.com/*
— PartCover : Appliquez l'éxécutable que vous trouverez à :
http ://github.com/sawilde/partcover.net4/downloads/

4.3.2 Avec une machine cliente avec visual studio 2012

Pour le bon fonctionnement de SonarRunner sous C#, il est nécessaire d'installer 1 plugin : FxCop 10.0. Pour cela, téléchargez **Microsoft Windows SDK for Windows 7 and .Net Framework 4**.

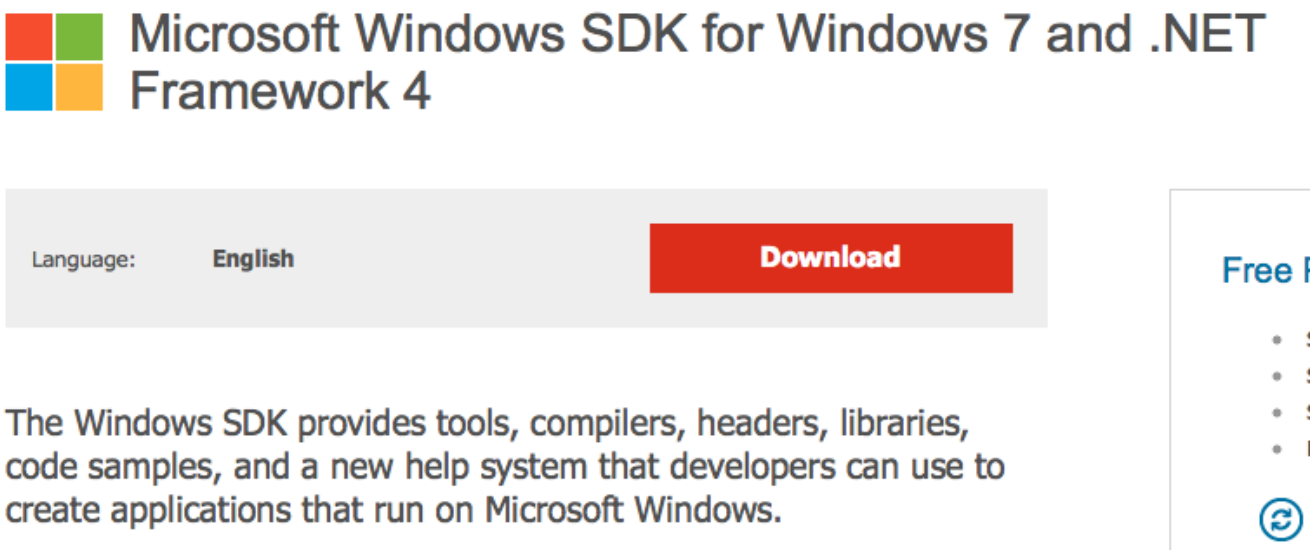

FIGURE 4.3 – Microsoft Windows SDK

Puis éxécutez l'éxcéutable FXCop que vous trouveriez dans le dossier textit C :/ProgramFiles/Microsoft SDKs/Windows/v7.1/Bin/FXCop

Conclusion

L'installation de la machine virtuelle est finie et opérationnelle. Comme vous avez pu le voir, peu de configurations sont finalement nécessaires pour l'utilisation du serveur. Cependant, afin de savoir l'utiliser correctement, vous pouvez vous référer au manuel utilisateur fourni dans l'ensemble de la documentation de ce projet de fin d'études.

Serveur d'Intégration Continue et Qualité de Code, côté client

Département Informatique
5e année
2013 - 2014

Manuel d'installation de la Machine Virtuelle Cliente

Résumé : Manuel d'installation des machines virtuelles clientes, afin de les rendre a-même d'utiliser le serveur d'intégration continue.

Mots clefs : Manuel d'Installation Client.

Abstract: Installation Manual

Keywords: Installation Manual

Encadrants
Nicolas Ragot
nicolas.monmarche@univ-tours.fr
Vincent T'Kindt
tkindt@univ-tours.fr

Étudiants
Anne CASTIER
anne.castier@etu.univ-tours.fr

DI5 2013 - 2014

Université François-Rabelais, Tours

École Polytechnique de l'Université de Tours
64, Avenue Jean Portalis
37200 TOURS, FRANCE
Tél. +33 (0)2 47 36 14 14
www.polytech.univ-tours.fr

Département Informatique
5e année
2013 - 2014

Manuel Utilisateur

Serveur d'Intégration Continue et Qualité de Code

Encadrants
Nicolas Ragot
nicolas.monmarche@univ-tours.fr
Vincent T'Kindt
tkindt@univ-tours.fr

Étudiants
Anne CASTIER
anne.castier@etu.univ-tours.fr

DI5 2013 - 2014

Université François-Rabelais, Tours

Version du 16 avril 2014

Table des matières

Introduction

Ce manuel utilisateur a pour but de vous aider à prendre en main de manière rapide et simple le serveur d'intégration continue et qualité de code construit durant mon projet de fin d'études. Si vous avez construit des projets C#, C/C++ avec Visual Studio 2010 ou 2012, ou encore des projets Java/Maven construits sous Eclipse, ce manuel est fait pour vous.
Bien entendu, si vous n'êtes pas dans ce cas là, ce manuel peut être quand même adapté à vos besoins (changements d'IDE, voir de langages), mais il se peut que vous soyez confrontés à quelques difficultés si vous utilisiez des IDEs ou langages plus exotiques... (Cf, dernière partie "Pour les autres langages").
Ce manuel se décompose en 3 parties : Projets Java Maven construits sous Eclipse, Projets C# construits sous Visual Studio 2012 ou 2010, Projets C/C++ construits sous Visual Studio 2010 ou 2012.

Projets Java Maven construits sous Eclipse

2.0.1 Préambule

Tout d'abord, il convient de répondre à une question que certains peuvent encore se poser : Pourquoi utiliser le moteur de production Maven ? Parce que, pour utiliser au mieux les capacités de Jenkins (notre serveur d'intégration continue) il est fortement conseillé de créer un projet Maven. Maven vous permettra d'automatiser certaines tâches, notamment de gérer les dépendances vis à vis des bibliothèques, ainsi que d'automatiser les tâches de compilation, tests unitaires et déploiement des applications.

Dans cette partie, nous recommandons l'utilisation de la machine virtuelle Windows 7 contenant Eclipse et préconfigurée pour l'utilisation de notre serveur d'intégration continue. Si vous préferez cependant avoir votre propre machine virtuelle, vous pouvez suivre le manuel d'installation (très rapide) de la VM cliente en partant d'une machine virtuelle de base (que vous pouvez trouver sur n'importe quelle machine de l'école dans le dossier D :/).

Enfin vous pouvez trouver dans le dossier "Projets" fourni avec ce projet l'exemple qui va être utilisé dans cette partie : **Projet Eclipse**.

2.0.2 Fonctionnement sous l'IDE

Avant toute chose, il faut être sûr que tout marche bien du côté de votre IDE. Lancez Eclipse (depuis C :/Program Files/Eclipse). Importez votre projet Maven (ou créez en-un). Dans notre cas, nous fonctionnons avec un projet déjà existant, on se contente juste de l'importer :

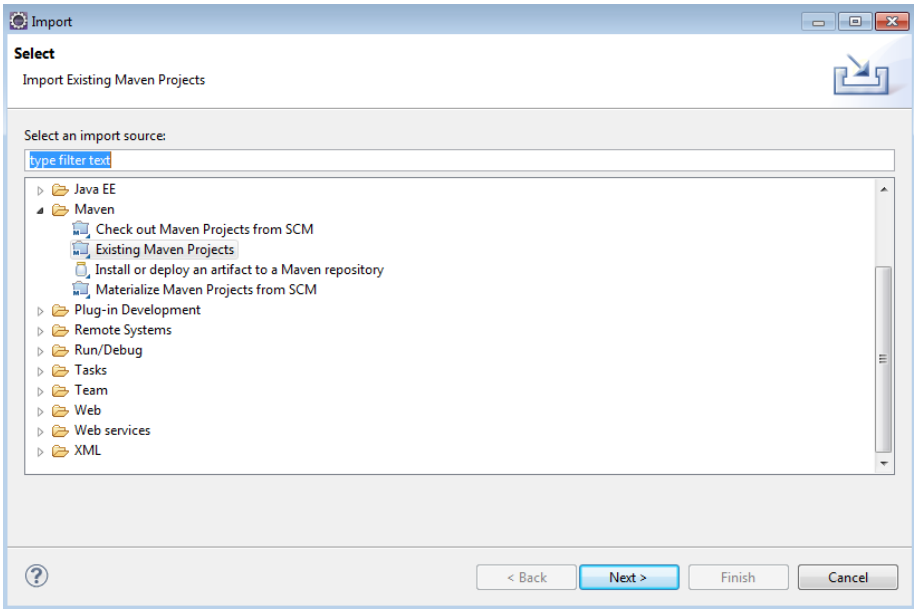

FIGURE 2.1 – Importation d'un projet existant

Sélectionnez ensuite le dossier contenant votre projet.
Une fois votre projet importé, compilez-le et lancez les tests unitaires.

```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running fr.jdev2013.user01.calculatrice.operator.SimpleOperatorsTest
sub
div
div1
add
Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.061 sec

Results :

Tests run: 4, Failures: 0, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 9.578s
[INFO] Finished at: Wed Mar 26 12:07:39 CET 2014
[INFO] Final Memory: 6M/15M
[INFO] ------------------------------------------------------------------------
```

FIGURE 2.2 – Compilation et tests unitaires

Si cette première étape se passe bien, vous pouvez passez à la suite.

2.0.3 Importation sous Redmine

Afin de pouvoir utiliser le serveur subversion fourni avec le redmine de l'école, il peut être utile (mais pas nécessaire) d'installer un logiciel client SVN. D'ailleurs, toutes les manipulations du manuel utilisateur seront faites à l'aide de ce client **TortoiseSVN** (déjà préinstallé si vous avez pris la VM cliente déjà préconfigurée).

En premier lieu, avant de commencer l'importation sous Redmine, vous aurez besoin d'une adresse de dépôt subversion. Pour cela allez à *http ://redmine.polytech.univ-tours.fr*, connectez vous avec vos identifiants Polytech.
Ensuite, deux cas s'offrent à vous :

1. Vous êtes en train de réaliser un projet dans le cadre de l'école et avez demandé la création d'un projet auprès du service informatique (démarche habituelle dans le cadre de PIL/PFE...). Dans ce cas, vous trouverez l'adresse de votre dépôt subversion dans l'onglet **Configuration**, onglet **Dépôts**.

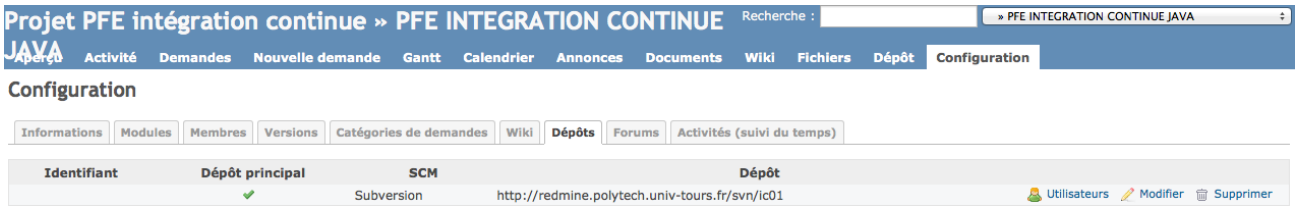

FIGURE 2.3 – Adresse du Dépôt SVN

2. Vous êtes dans le cadre d'un TP pour tester le serveur d'intégration continue. Dans ce cas-là, vous pouvez créer un projet simple (en respectant les conventions de nommage de polytech *ANNEECIVILE_PROMO_NOMPROJET* ou *ANNEECIVILE_PROMO_TP_NřETU*). Pour cela, allez dans Projets, cliquez sur Nouveau Projet.

.

FIGURE 2.4 – Adresse du Dépôt SVN, création d'un projet, étape 1

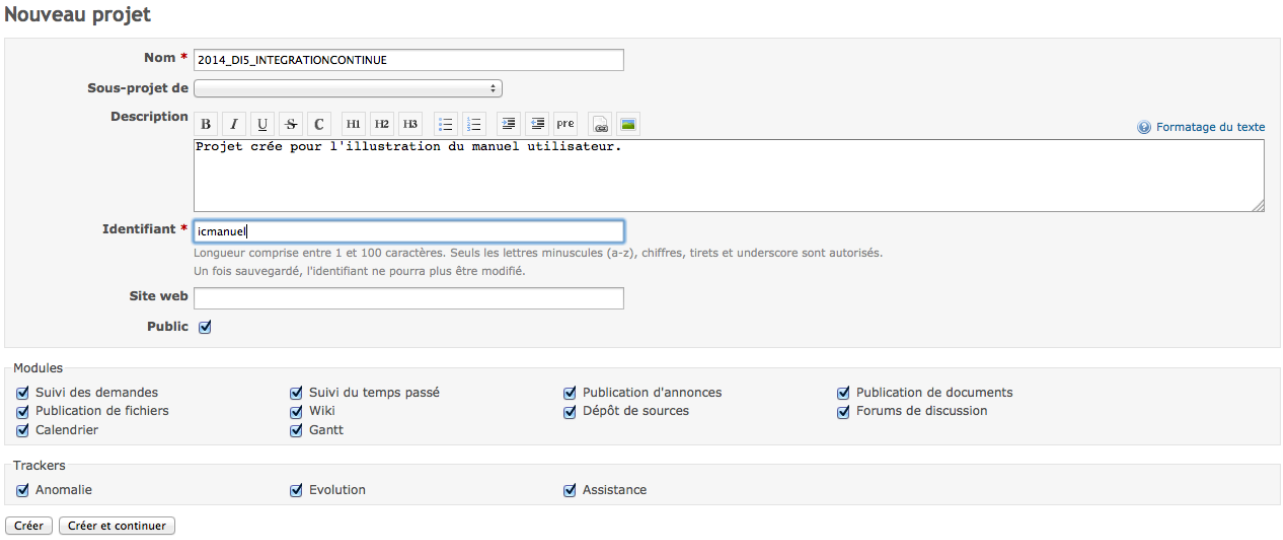

FIGURE 2.5 – Adresse du Dépôt SVN, création d'un projet, étape 2

Ensuite, vous pouvez récupérer l'adresse de votre dépôt subversion dans l'onglet **Configuration**, onglet **Dépôts**.

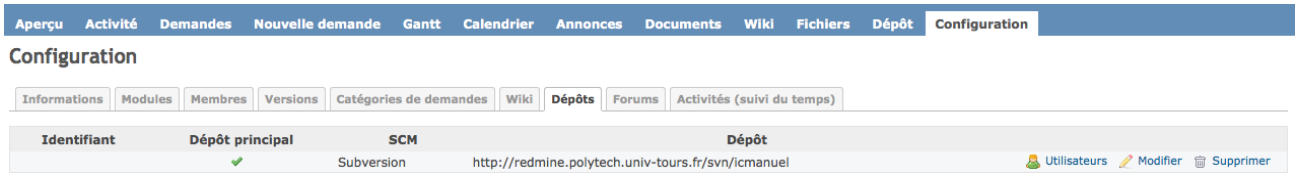

FIGURE 2.6 – Adresse du Dépôt SVN

.

Une fois l'adresse du dépôt subversion en votre possession, vous pouvez retourner auprès de votre projet. Créez un dossier "ProjetServeurIC". Dans ce dossier, créez 3 dossiers trunk, branches, tags. Placez le projet entier dans le dossier trunk.
Dans notre cas voici à quoi cela ressemble :

.

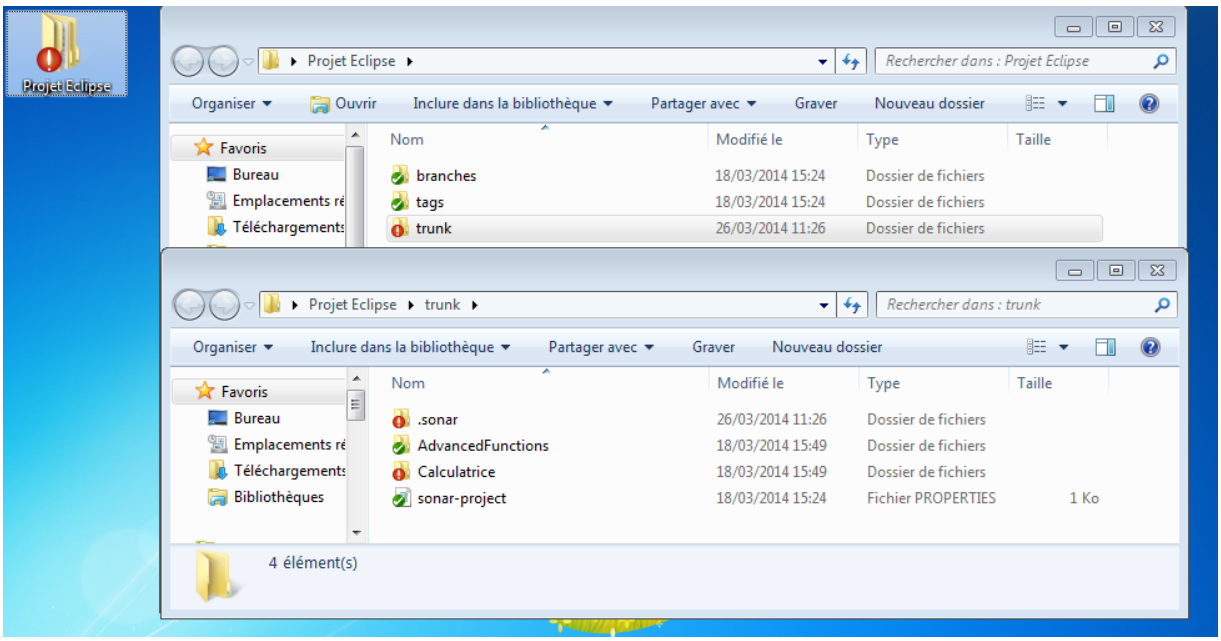

FIGURE 2.7 – Architecture des dossiers pour SVN.

.

Cliquez droit sur ce **projet** > **SVNTortoise** > **Import**. Rentrez l'adresse du dépôt subversion que vous venez d'obtenir, et n'oubliez pas d'ajouter un message d'importation (*c'est une bonne pratique à adopter !*).

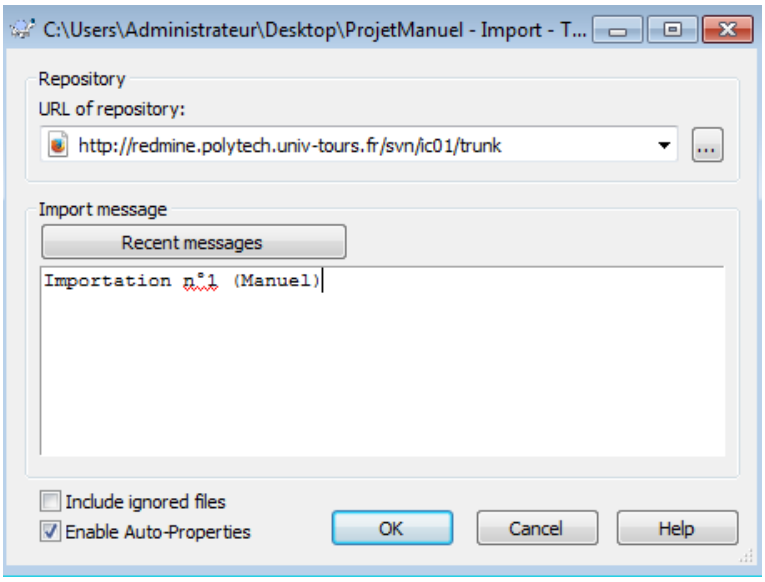

FIGURE 2.8 – Première importation.

Vous pouvez bien sûr aller vérifier sur redmine le bon fonctionnement de cette dernière opération. Allez à *http ://redmine.polytech.univ-tours.fr*, identifiez-vous si nécessaire, allez sur votre projet, onglet **Dépôt**.

Ça y est, votre projet est dès à présent sur votre dépôt subversion. On va donc dès à présent pouvoir attaquer la partie plus intéressante : Jenkins.

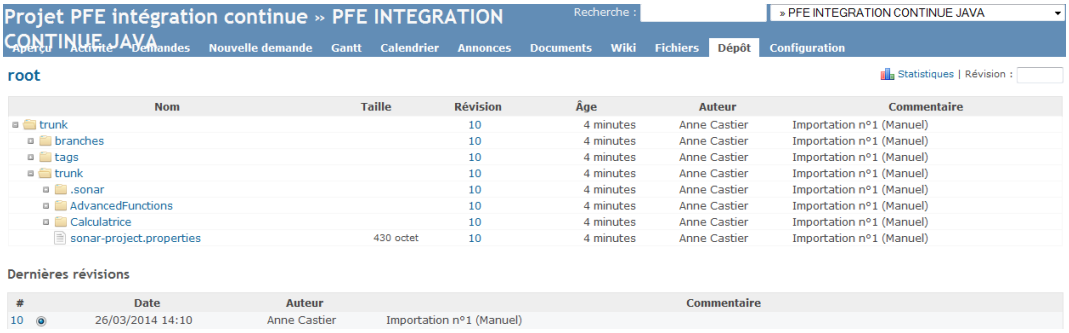

FIGURE 2.9 – Verification Importation

2.0.4 Création du Job sous Jenkins

Grâce à l'adresse que l'on vous a fourni dans le cadre de ce TP, connectez vous à Jenkins à l'adresse *adresseserveur :8080/jenkins*. Il vous sera demandé de vous identifier, chose que vous pouvez faire naturellement avec vos identifiants étudiants (numéro étudiant suivi de t / password). Si vous ne disposez pas de droits d'accès supplémentaires, rendez vous auprès de l'équipe du service informatique, avec le nom de votre groupe (que vous pouvez connaître à l'adresse *adresseserveur :8080/jenkins/whoAmI*.

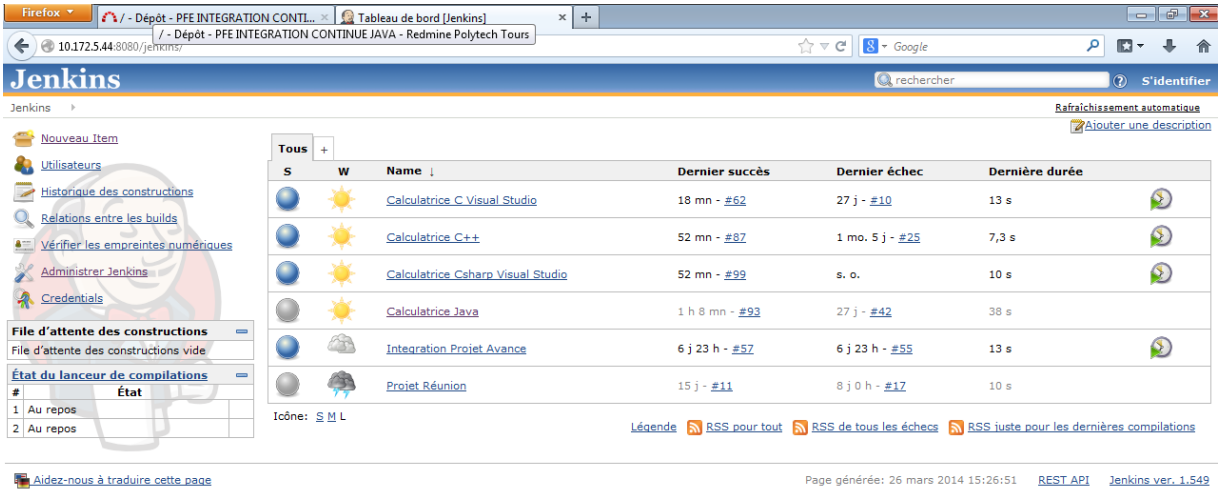

FIGURE 2.10 – Jenkins, page d'accueuil

Voici les différentes étapes de configuration d'un job sous Jenkins pour un projet Java/Maven.

1. **Création :** Cliquez sur Nouveau Item, puis cochez l'option construire un projet maven 2/3 et donnez un nom (sans caractères exotiques...) à votre job.

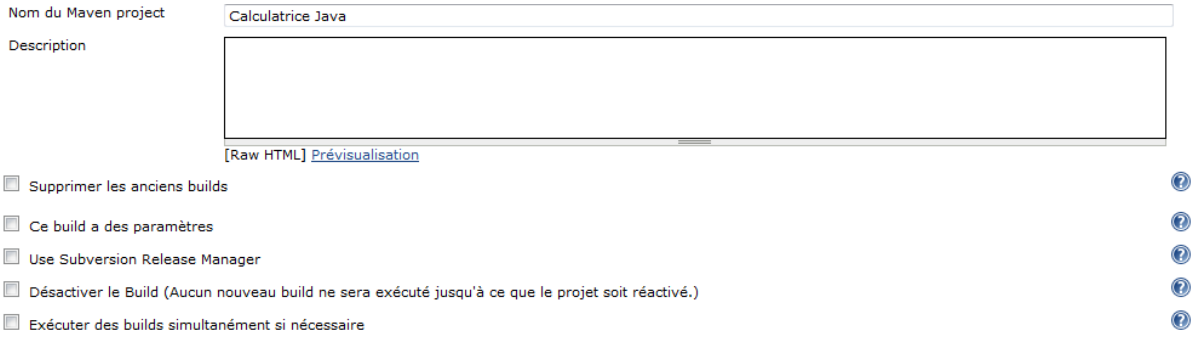

FIGURE 2.11 – Jenkins,Configuration étape 1

Bien sûr, il est fortement déconseillé de laisser l'accès à d'autres personnes. De ce fait, n'oubliez pas de vous attribuer tous les droits sur votre propre projets à vous ainsi qu'aux autres personnes/groupes impliquées dans le projet.

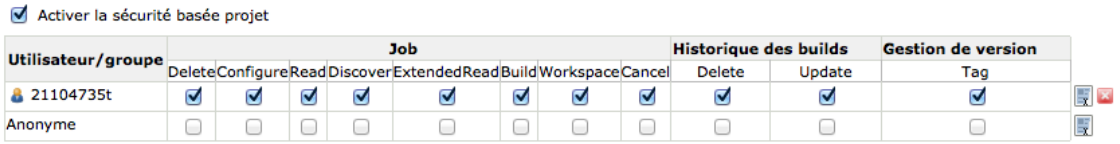

FIGURE 2.12 – Jenkins, sécurité.

2. **Configuration, étape 1 :** Vous pouvez bien évidemment configurer le projet selon votre envie, mais au début, il vaut mieux aller au plus court. Dans la partie gestion de code source, assure vous que le repository URL pointe vers le dossier contenant le pom.xml qui vous intéresse (ici trunk/trunk/Calculatrice). Il vous sera demandé vos "credentials" à un moment : ce sont vos identifiants de l'école (numérot étudiant/mot de passe associé).

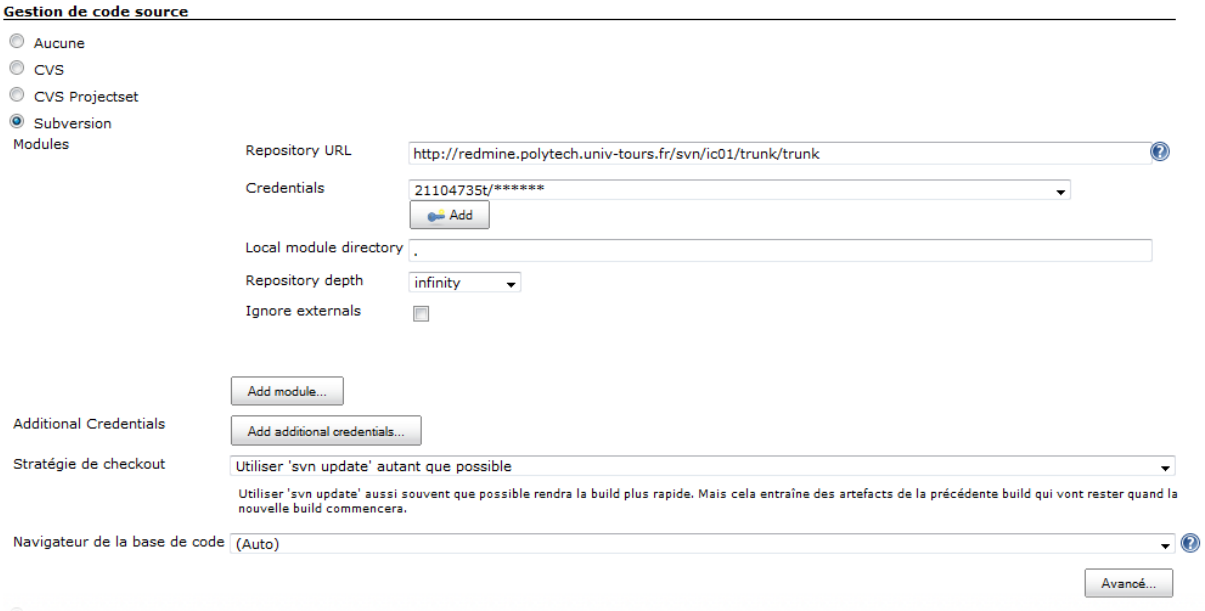

FIGURE 2.13 – Jenkins, Configuration étape 2

3. **Configuration, étape 2 :** Vous pouvez ensuite commander à jenkins, d'éxécuter votre build conti-

nuellement (et c'est tout l'intérêt) ! Ici on demande à Jenkins de construire notre build toutes les heures mais AUSSI de scruter l'outil de gestion de version toutes les minutes. Ainsi à chaque commit d'un développeur travaillant sur le projet, Jenkins va lancer un build et les tests allant avec ! Ainsi à la moindre régression de code, tous les développeurs seront mis au courant en consultant l'interface web de Jenkins.

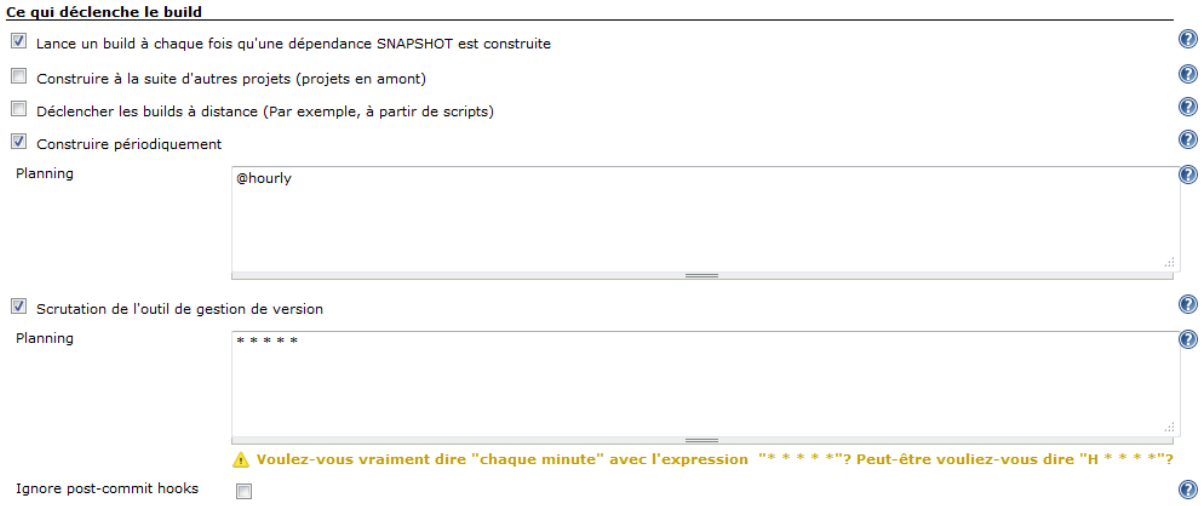

FIGURE 2.14 – Jenkins, Configuration étape 3

4. **Configuration, étape 3 :** Indiquez juste le fichier pom utilisé (ici pom.xml) dans l'étape build. Cliquez sur le bouton Sauver et lancez un build, pour voir si tout fonctionne.

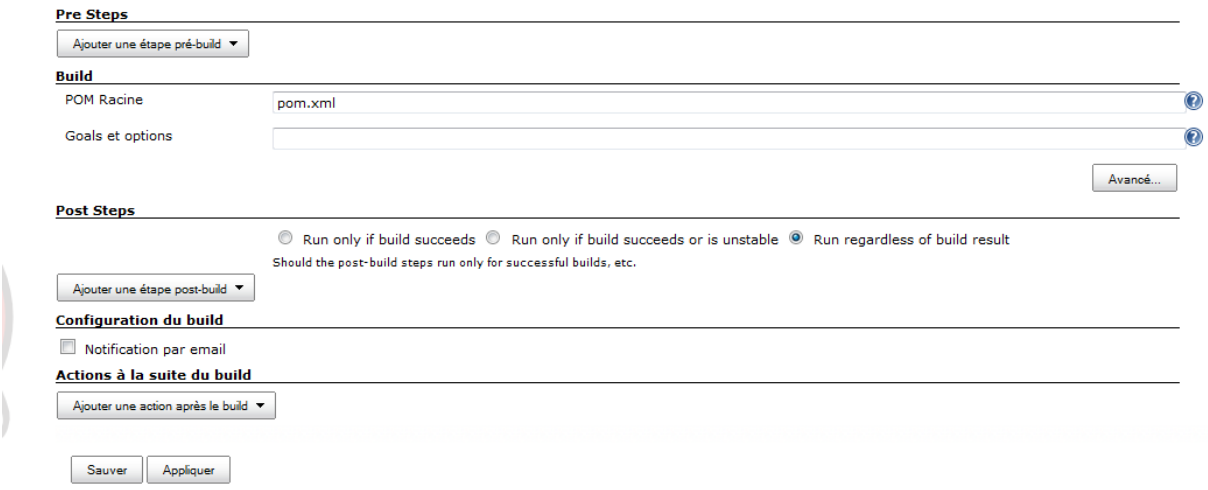

FIGURE 2.15 – Jenkins, Configuration étape 4

5. **Lancement du premier build :** Clickez, sur l'onglet en haut à gauche : **"Lancer un build"**. Pour être sûr et certain que tout fonctionne bien vous pouvez accèder à la sortie de la console, en cliquant sur le build en train de s'éxécuter puis sur sortie console.

FIGURE 2.16 – Jenkins, Sortie Console

Voilà, la configuration du job Jenkins est finie ! Cependant, on va tester dans la partie suivante sa réelle efficacité...

2.0.5 Une petite vérification s'impose

Testons la réelle aptitude de Jenkins à détecter les régressions de code... *Imaginez l'un de vos coéquipiers récupérer le code source, changer le code source et de ce fait faire échouer un test qui réussissait auparavant. Ce coéquipier, sans se rendre compte de son erreur, va commiter son changement.* **Jenkins va-t-il détecter automatiquement cette régréssion ?**.

Pour cela, plusieurs étapes sont nécéssaires :

1. Mettez vous à la place de votre coéquipiez et récupèrez le code source depuis Redmine. Pour cela click droit sur votre bureau > Checkout. Indiquez l'adresse de votre dépôt SVN (comme dans la première étape).

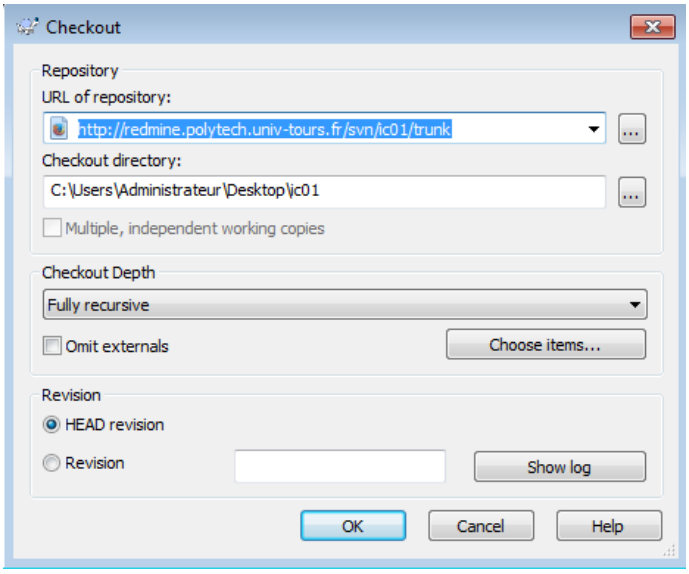

FIGURE 2.17 – Premier CheckOut

2. Comme tout à l'heure, importez votre projet dans Eclipse. Modifiez l'un des tests de manière à ce que celui-ci échoue (dans mon cas j'ai modifié le test testAdd, en changeant la valeau de number1 par 9.0). Sauvegardez et fermez eclipse.

3. Click droit sur le dossier récupéré à la première étape, puis cliquez sur SVN Commit.

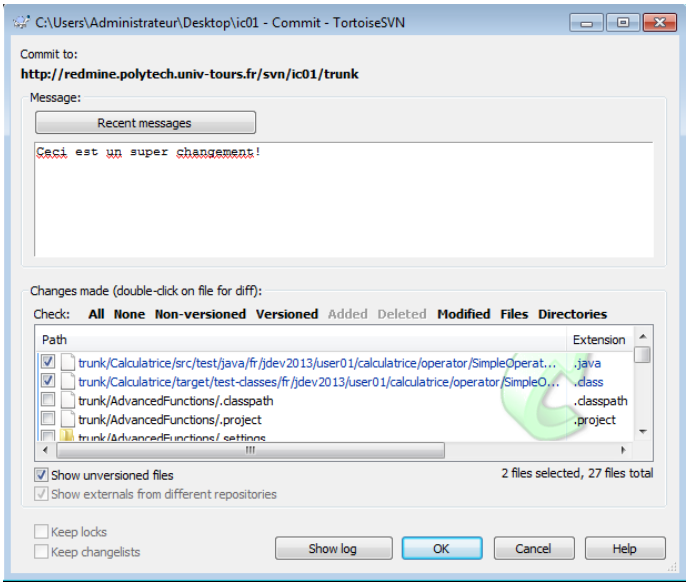

FIGURE 2.18 – C'est un super changement !

4. Enfin retournez sur votre serveur jenkins. Attendez une petite minute, et regardez votre build s'éxécuter. Si tout s'est "bien" passé, Jenkins aura remarqué le changement. L'icône du build a viré jaune, et le graphique de résultats de tests a tourné au rouge... .

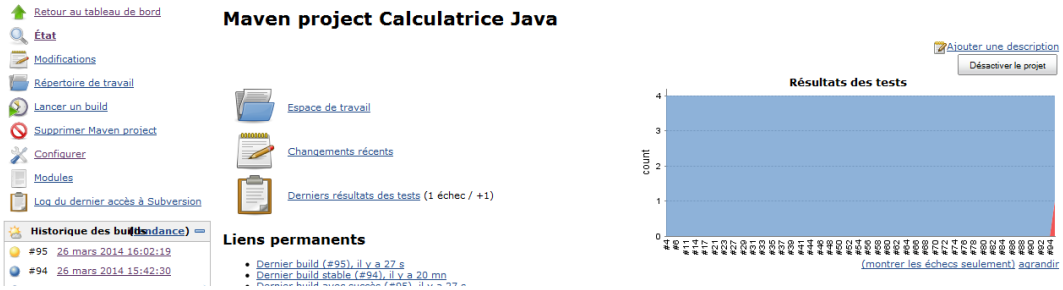

FIGURE 2.19 – Un probleme ?

En cliquant sur le build, vous aurez plus d'informations : quel test à échoué, quand a eu lieu le commit, et vous pourrez même savoir à qui est dûe cette erreur !

2.0.6 Lancement d'une analyse de qualité de code avec SonarRunner

Avant tout chose, comparez l'adresse du serveur que l'on vous aura préalablement fournie avec l'adresse située dans le dossier c :/Program Files/SonarRunner/conf . Si celles-ci sont différentes, remplacez l'adresse ip 10.172.5.44 (que vous pouvez voir sur le screenshot) par celle que l'on vous a fournie. Il vous faudra à nouveau vous connecter avec vos identifiants habituels (numéro étudiant suivi de t, et mot de passe associé).

```
#----- Default SonarQube server
sonar.host.url=http://10.172.5.44:8080/sonar

#----- PostgreSQL
sonar.jdbc.url=jdbc:postgresql://10.172.5.44:5432/sonar
sonar.jdbc.driverClassName=org.postgresql.Driver
```

FIGURE 2.20 – Configuration SonarRunner

Puis, rendez vous à la racine de votre projet et créez un fichier texte sonar-project.properties.

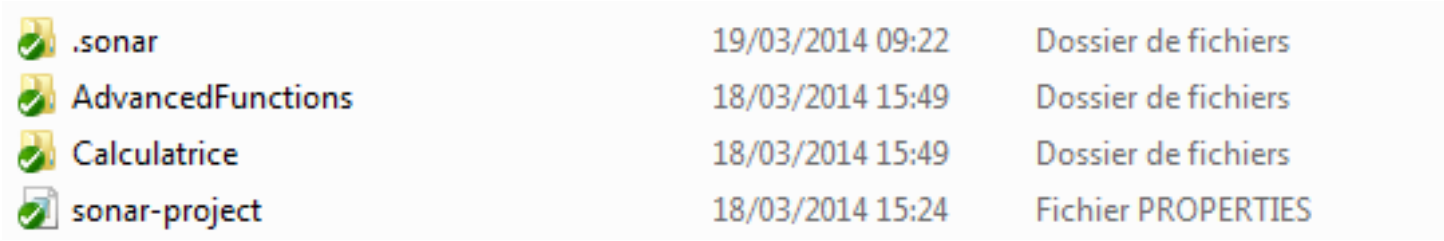

FIGURE 2.21 – Architecture du Projet

Remplissez le fichier sonar-project.properties de la manière suivante.

1. Identification du projet :
   - **sonar.projectKey=my :project**
   - **sonar.projectName=My project**
   - **sonar.projectVersion=1.0**
2. Identification des sources :
   On y indique le chemin du repertoire parent contenant les sources (en général **src**). Dans l'exemple présenté dans l'image suivante, comme le projet est divisé en deux sous projets (AdvancedFunctions et Calculatrice) il faut le préciser en ajoutant sonar.modules.
   - **sonar.sources=src**
   - **sonar.modules=module1,module2** (optionnel).
   - **module1.sonar.projectName=module1** (optionnel).
3. Précision du langage
   - **sonar.language=java**

.

NB : Les commentaires sont précédés par le caractère#.
NB 2 : Vous pouvez trouver plus d'informations à l'adresse
http ://docs.codehaus.org/display/SONAR/Analysis+Parameters

```
# Project identification
sonar.projectKey=Sample:CalculatriceJava
sonar.projectVersion=1.0
sonar.projectName=Java-Calculatrice

# Info required for Sonar
sources=src/main/java
sonar.modules=Calculatrice, AdvancedFunctions
Calculatrice.sonar.projectName = Calculatrice
AdvancedFunctions.sonar.projectName = AdvancedFunctions
sonar.language=java
sonar.scm.url=scm:svn:http://redmine.polytech.univ-tours.fr/svn/ic01/
```

FIGURE 2.22 – Configuration du fichier sonar-project

Voilà, la configuration est finie.
On va donc pouvoir lancer une analyse à l'aide de Sonar-Runner. Pour cela rendez vous, depuis une invite de comandes, dans le dossier contenant le fichier sonar-project.properties. Lancez la commande sonar-runner, suivi de -Dsonar.login=numEtudiant -Dsonar.password=votremotdepasse.

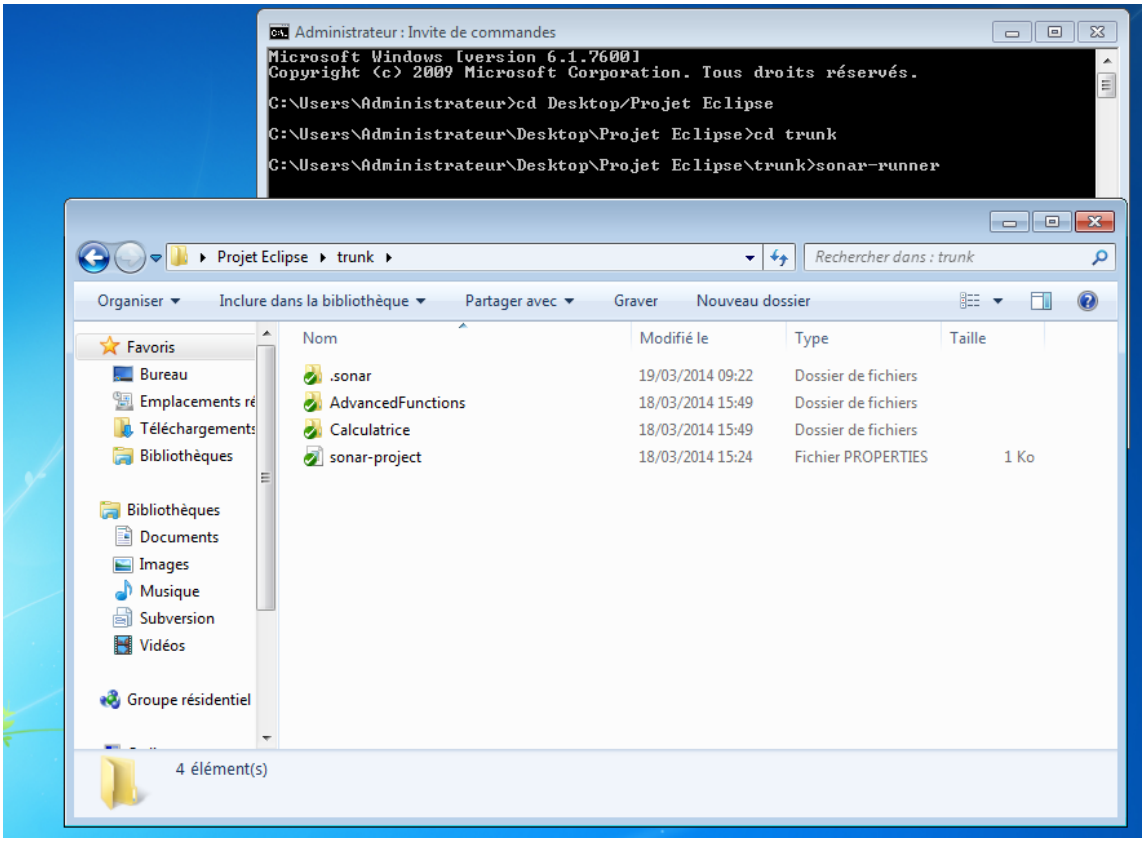

FIGURE 2.23 – Lancement de l'analyse

Si tout se passe bien, vous devriez pouvoir voir l'écran suivant :

```
INFO: ------------------------------------------------------------------------
INFO: EXECUTION SUCCESS
INFO: ------------------------------------------------------------------------
Total time: 2:22.750s
Final Memory: 12M/113M
INFO: ------------------------------------------------------------------------
```

FIGURE 2.24 – Réussite de l'analyse

Enfin, vous pouvez accèder à l'outil SonarQube à l'adresse de votre serveur d'intégration continue suivi par *:8080/sonar*. Cliquez sur votre projet, voici ce que vous devriez avoir obtenu.

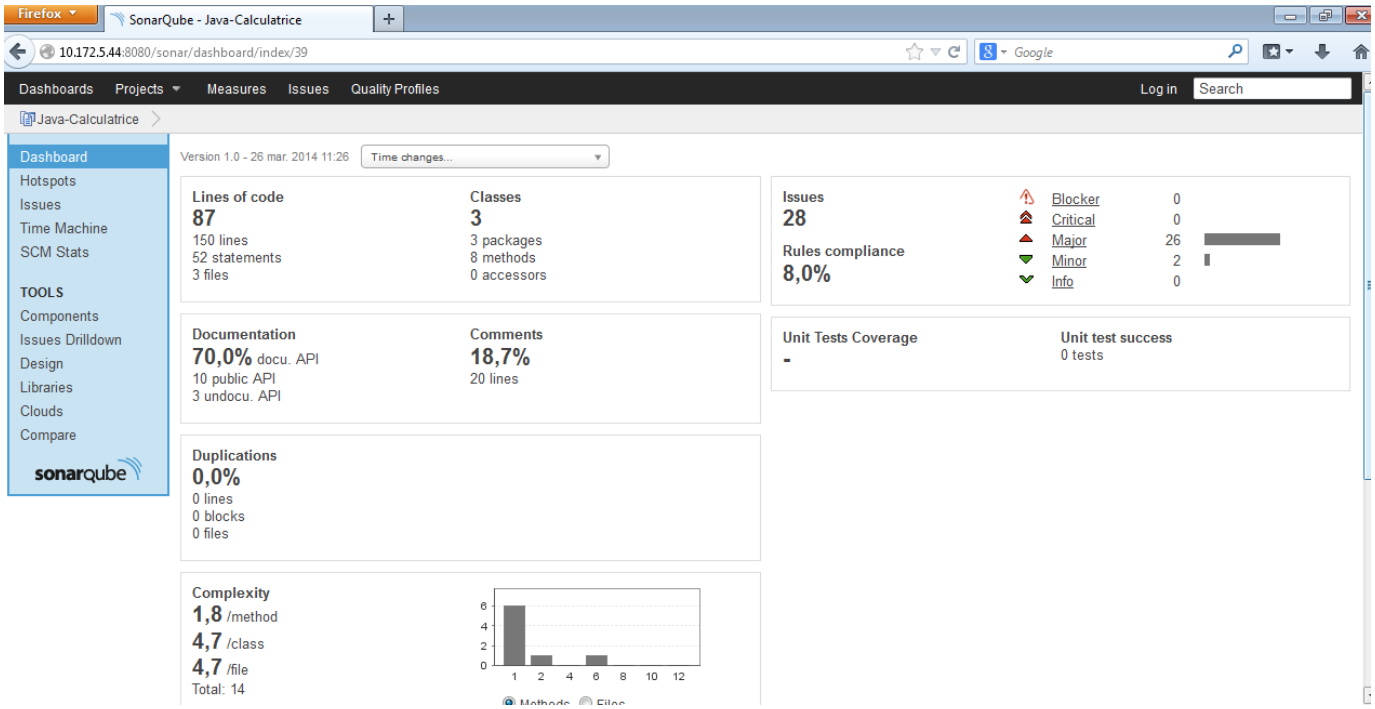

FIGURE 2.25 – Interface SonarQube

Il ne vous reste plus qu'à découvrir SonarQube... !

2.0.7 Lancement d'une analyse de qualité de code depuis Jenkins

Jenkins et SonarQube étant fortement orienté Java, il existe un plugin permettant de lancer une analyse sur SonarQube à l'aide de Maven après un build. Cette analyse est extrêmement rapide à configurer.

Depuis la configuration de votre job sur l'interface de Jenkins, rajouter une étape supplémentaire :

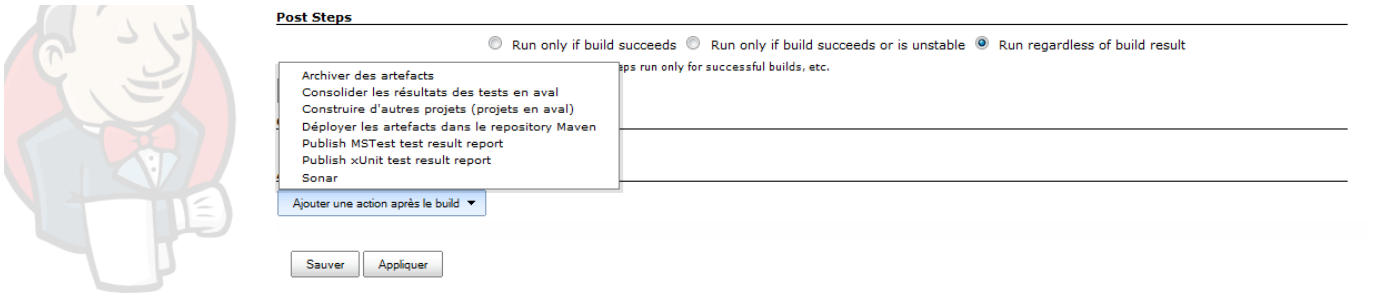

FIGURE 2.26 – Configuration du build pour lancer une analyse sonar

Sauvegardez puis lancez votre build. Vous devriez voir apparaître dans la sortie console le message suivant à la fin du build :

```
Sonar analysis completed: SUCCESS
Finished: SUCCESS
```

FIGURE 2.27 – Réussite du build

Enfin, vous pouvez accèder à l'outil SonarQube à l'adresse de votre serveur d'intégration continue suivi par *:8080/sonar*. Cliquez sur votre projet, vous devriez obtenir à peu près le même résultat que pour le lancement d'analyse avec SonarRunner.

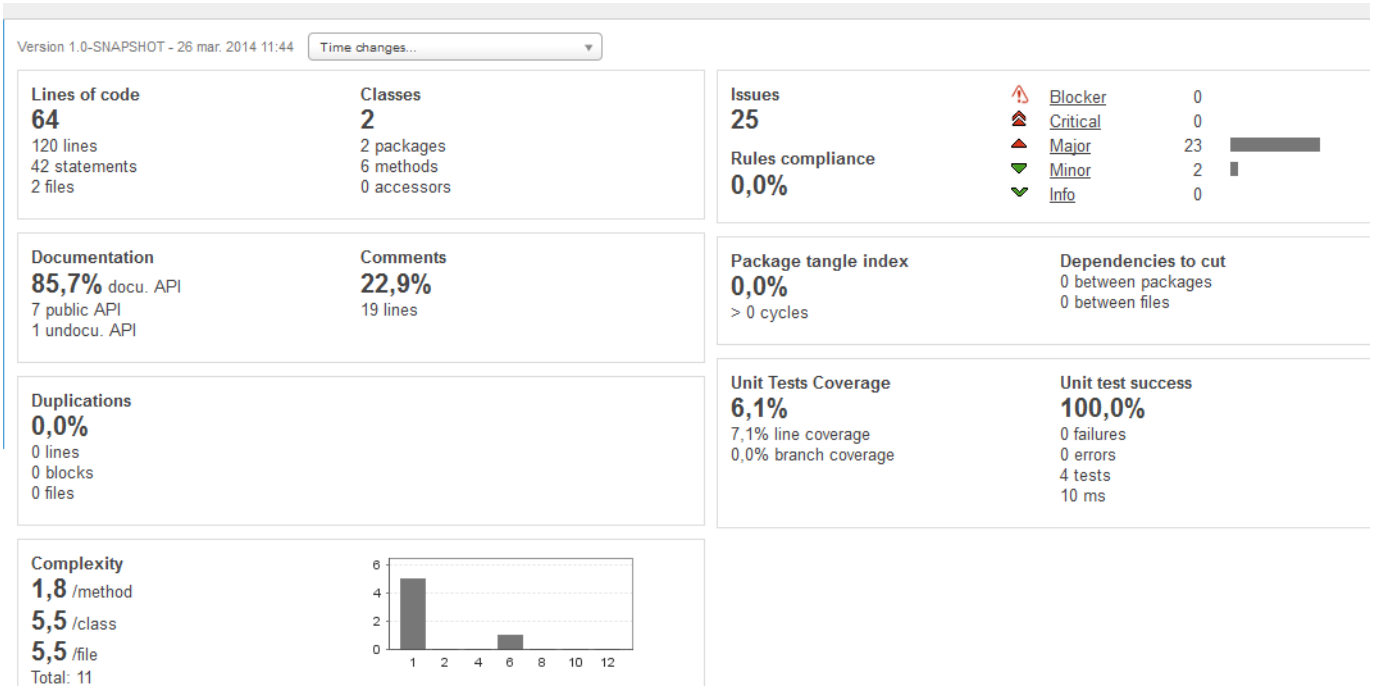

FIGURE 2.28 – Résultats depuis SonarQube

Cependant, lancer une analyse avec SonarRunner est plus conseillée puisque plus complète (les rules compliances n'ont pas été appliquées dans ce deuxième exemple, comme vous pouvez le voir elles sont à 0%).

Projets C# construits sous Visual Studio 2010 ou 2012

3.0.8 Préambule

Dans le cadre de projets C# développés sous Visual Studio 2010 ou 2012, il est également conseillé d'utiliser l'une des machines clientes construites durant ce projet de fin d'études et préconfigurée pour l'utilisation du serveur d'intégration continue et qualité de code. L'une des machines virtuelles clientes contient Visual Studio 2010, l'autre Visual Studio 2012.
Si vous préferez cependant avoir votre propre machine virtuelle, vous pouvez suivre le manuel d'installation (très rapide) de la VM cliente en partant d'une machine virtuelle de base (que vous pouvez trouver sur n'importe quelle machine de l'école dans le dossier D :/).

Enfin vous pouvez trouver dans le dossier "Projets" fourni avec ce projet l'exemple qui va être utilisé dans cette partie : **Projet C#**.

3.0.9 Fonctionnement sous l'IDE

Avant toute chose, il faut être sûr que tout marche bien du côté de votre IDE.
Pour cela plusieurs étapes sont nécéssaires :

— Lancez votre solution .sln avec l'IDE souhaité (Visual Studio 2010 ou 2012).
— Générez le projet. (Cliquez sur **Générer**, puis **Générer la solution**)
— Exécutez-le (Cliquez sur **Déboguer**, puis sur **Executez sans débogage**).

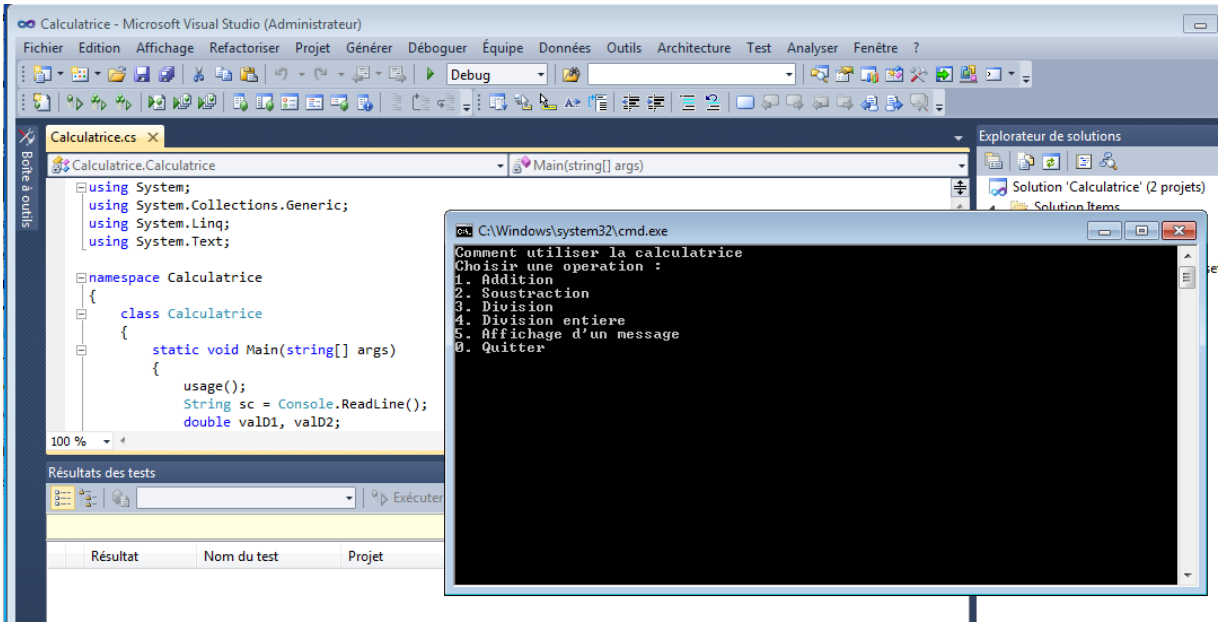

FIGURE 3.1 – Tests en ligne de commande

— Vérifiez que l'ensemble des tests s'éxécute bien (Cliquez sur **Test**, **Exécuter**, **Tous les tests de la solution**).

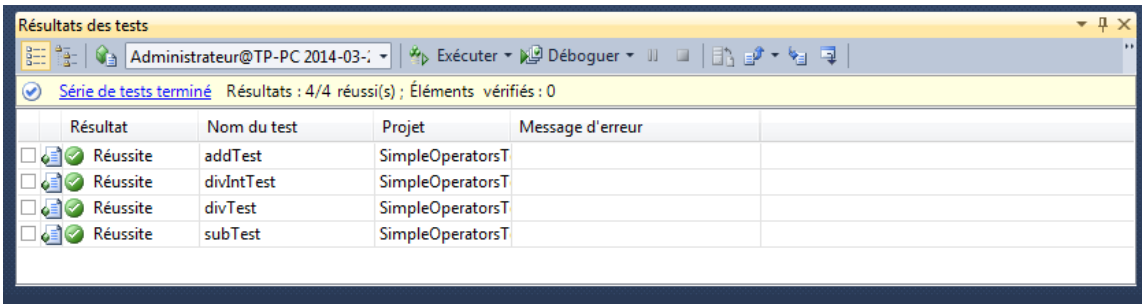

FIGURE 3.2 – Tests en ligne de commande

Voilà, votre projet est bel et bien fonctionnel sous votre IDE ! On peut donc passer à la suite et attaquer le travail sur le serveur d'intégration continue.

Attention : Une autre étape est également nécessaire. Pour avoir vos tests fonctionnant sous Jenkins, ceux-ci doivent s'éxécuter en lignes de commandes avec MsTest. Cela peut paraître anodin, mais si ce n'est pas le cas, ça ne fonctionnera pas non plus sous Jenkins.

1. Ainsi, pour vérifier le bon fonctionnement des tests en lignes de commande, allez dans le menu **Démarrer**, pointez sur **Tous les programmes**, sur **Microsoft Visual Studio 2012**, sur **Visual Studio Tools**, puis cliquez sur **Invite de commandes développeur**.
2. Pour lancer vos tests, utilisez la commande MsTest. Vous spécifierez le fichier de tests à lancer (en .dll, que vous trouverez dans le dossier bin/Debug du projet de Test, *dans notre cas*) à l'aide de l'option /testcontainer.

```
C:\Program Files\Microsoft Visual Studio 10.0\VC>MsTest /testcontainer:"C:/Users
/Administrateur/Desktop/Projets VS 2010/csharp/trunk/ConsoleApplication1/SimpleO
peratorsTest/bin/Debug/SimpleOperatorsTest.dll"
Microsoft (R) Test Execution Command Line Tool Version 10.0.30319.1
Copyright (C) Microsoft Corporation 2005. Tous droits réservés.

Chargement en cours de C:\Users\Administrateur\Desktop\Projets VS 2010\csharp\tr
unk\ConsoleApplication1\SimpleOperatorsTest\bin\Debug\SimpleOperatorsTest.dll...

Démarrage de l'exécution...

Résultats                       Tests de niveau supérieur
---------                       -------------------------
Réussite                        SimpleOperatorsTest.SimpleOperatorsTest.addTest
Réussite                        SimpleOperatorsTest.SimpleOperatorsTest.divIntTes
t
Réussite                        SimpleOperatorsTest.SimpleOperatorsTest.divTest
Réussite                        SimpleOperatorsTest.SimpleOperatorsTest.subTest
4/4 test(s) Réussite

Résumé
------
Série de tests Terminé.
  Réussite  4
  ----------
  Total     4
Fichier de résultats : C:\Program Files\Microsoft Visual Studio 10.0\VC\TestResu
lts\Administrateur_TP-PC 2014-03-27 09_17_56.trx
Paramètres de test :   Paramètres de test par défaut

C:\Program Files\Microsoft Visual Studio 10.0\VC>
```

FIGURE 3.3 – Tests en ligne de commande

3. Vous pouvez évidemment spécifiez plus d'options selon les besoins de votre projet. Vous trouverez plus d'informations sur *http ://msdn.microsoft.com/fr-fr/library/ms182489.aspx*.

3.0.10 Importation sous Redmine

Afin de pouvoir utiliser le serveur subversion fourni avec le redmine de l'école, il peut être utile (mais pas nécessaire) d'installer un logiciel client SVN. D'ailleurs, toutes les manipulations du manuel utilisateur seront faites à l'aide de ce client TortoiseSVN (déjà préinstallé si vous avez pris la VM cliente déjà préconfigurée).

En premier lieu, avant de commencer l'importation sous Redmine, vous aurez besoin d'une adresse de dépôt subversion. Pour cela allez à *http ://redmine.polytech.univ-tours.fr*, connectez vous avec vos identifiants Polytech.

Ensuite, deux cas s'offrent à vous :

1. Vous êtes en train de réaliser un projet dans le cadre de l'école et avez demandé la création d'un projet auprès du service informatique (démarche habituelle dans le cadre de PIL/PFE...). Dans ce cas, vous trouverez l'adresse de votre dépôt subversion dans l'onglet **Configuration**, onglet **Dépôts**.

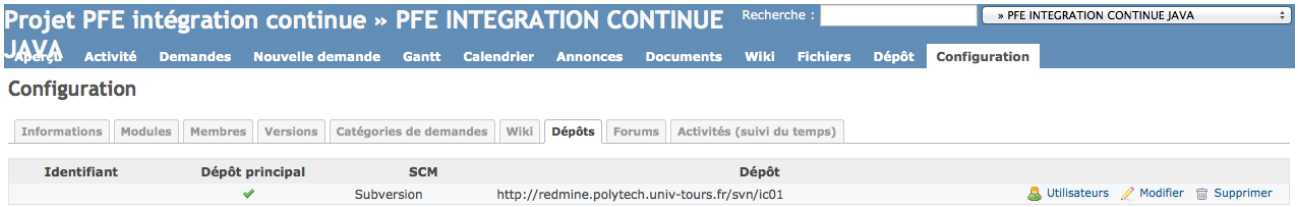

FIGURE 3.4 – Adresse du Dépôt SVN

2. Vous êtes dans le cadre d'un TP pour tester le serveur d'intégration continue. Dans ce cas-là, vous pouvez créer un projet simple (en respectant les conventions de nommage de polytech *ANNEECIVILE_PROMO_NOMPROJET* ou *ANNEECIVILE_PROMO_TP_NřETU*). Pour cela, allez dans Projets, cliquez sur Nouveau Projet.

FIGURE 3.5 – Adresse du Dépôt SVN, création d'un projet, étape 1

Puis remplissez les informations nécéssaires et demandées pour ce projet (Nom, description, identifiant...)

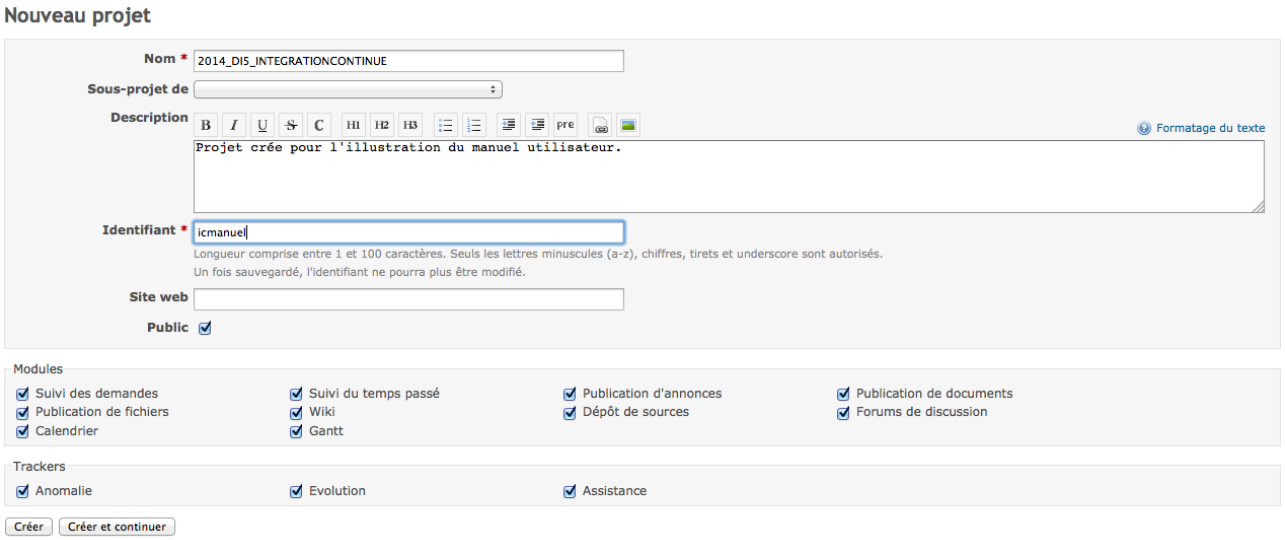

FIGURE 3.6 – Adresse du Dépôt SVN, création d'un projet, étape 2

Ensuite, vous pouvez récupérer l'adresse de votre dépôt subversion dans l'onglet **Configuration**, onglet **Dépôts**.

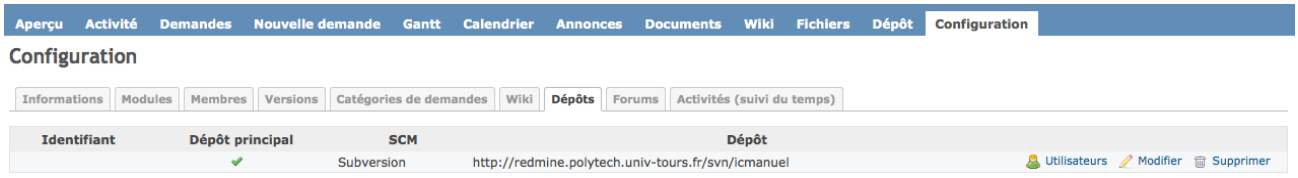

FIGURE 3.7 – Adresse du Dépôt SVN

Une fois l'adresse du dépôt subversion en votre possession, vous pouvez retourner auprès de votre projet. Créez un dossier "NomduProjet". Dans ce dossier, créez 3 dossiers trunk, branches, tags. Placez le projet entier dans le dossier trunk.

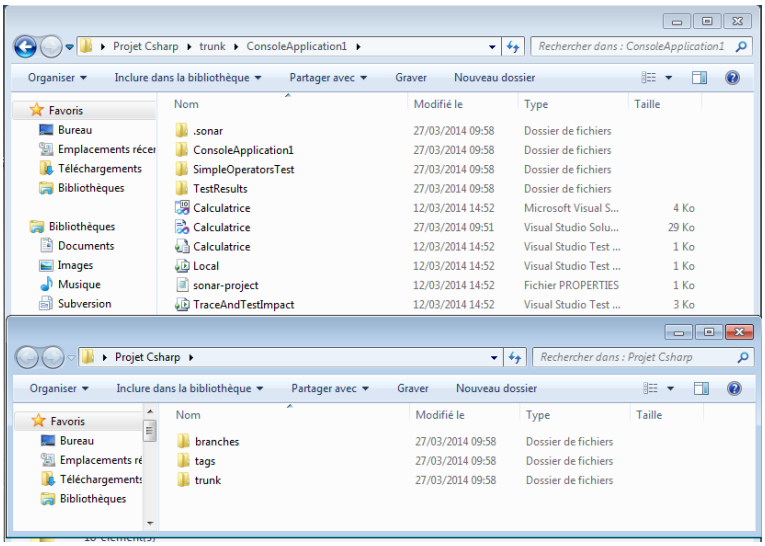

FIGURE 3.8 – Architecture des dossiers pour SVN.

Cliquez droit sur ce **projet** > **SVNTortoise** > **Import**. Rentrez l'adresse du dépôt subversion que vous venez d'obtenir, et n'oubliez pas d'ajouter un message d'importation (*c'est une bonne pratique à adopter !*).

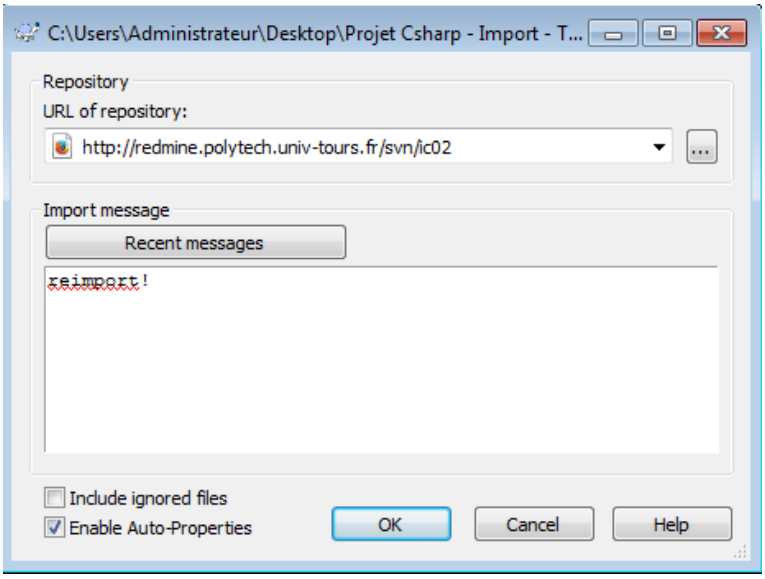

FIGURE 3.9 – Première importation.

Vous pouvez bien sûr aller vérifier sur redmine le bon fonctionnement de cette dernière opération. Allez à *http ://redmine.polytech.univ-tours.fr*, identifiez-vous si nécessaire, allez sur votre projet, onglet **Dépôt**.

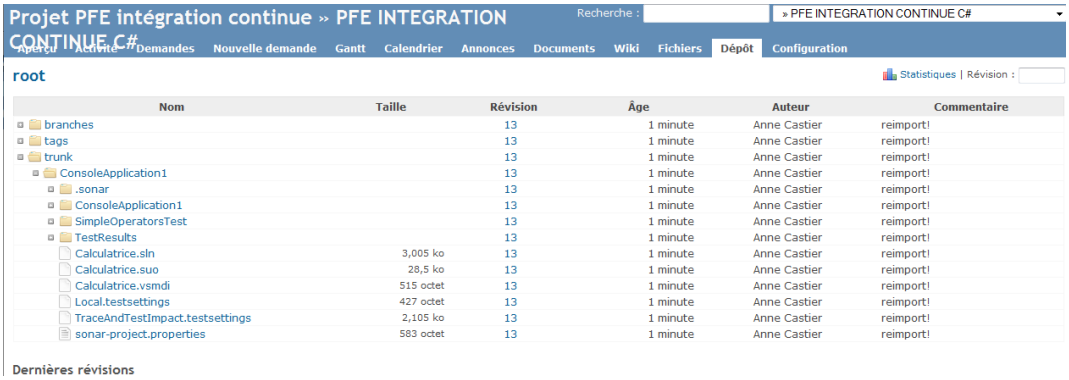

FIGURE 3.10 – Verification Importation

Ça y est, votre projet est dès à présent sur votre dépôt subversion. On va donc dès à présent pouvoir attaquer la partie plus intéressante : Jenkins.

3.0.11 Création du Job sous Jenkins

Grâce à l'adresse que l'on vous a fourni dans le cadre de ce TP, connectez vous à Jenkins à l'adresse *adresseserveur :8080/jenkins*. Il vous sera demandé de vous identifier, chose que vous pouvez faire naturellement avec vos identifiants étudiants (numéro étudiant suivi de t / password). Si vous ne disposez pas de droits d'accès supplémentaires, rendez vous auprès de l'équipe du service informatique, avec le nom de votre groupe (que vous pouvez connaître à l'adresse *adresseserveur :8080/jenkins/whoAmI*.

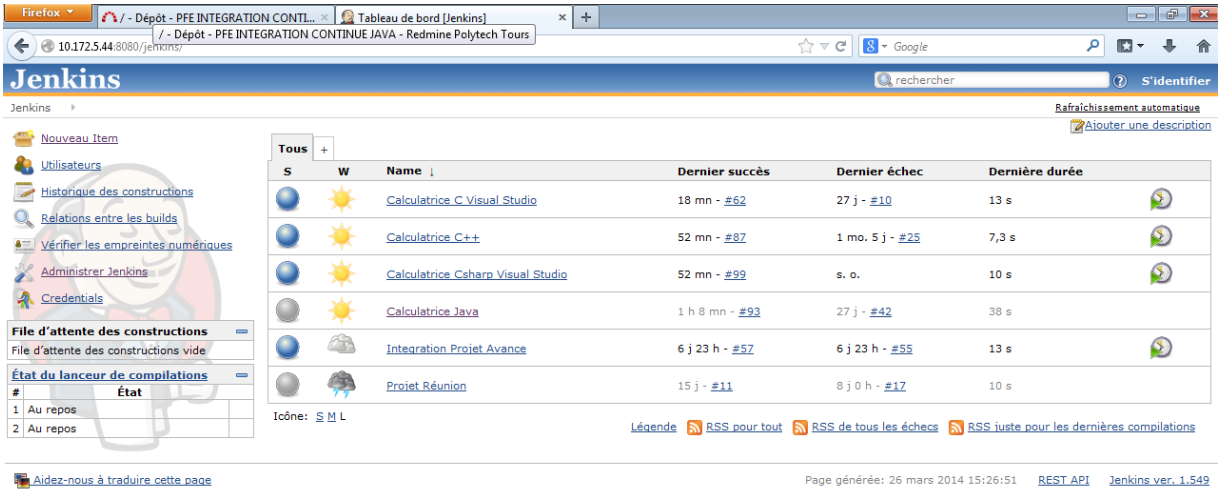

FIGURE 3.11 – Jenkins, page d'accueil

Voici les différentes étapes de configuration d'un job sous Jenkins pour un projet C# Visual Studio .

1. **Création :** Cliquez sur Nouveau Item, puis cochez l'option construire un projet freestyle et donnez un nom (sans caractères exotiques...) à votre job.

FIGURE 3.12 – Jenkins, Configuration étape 1

Bien sûr, il est fortement déconseillé de laisser l'accès à d'autres personnes. De ce fait, n'oubliez pas de vous attribuer tous les droits sur votre propre projets à vous ainsi qu'aux autres personnes/groupes impliquées dans le projet.

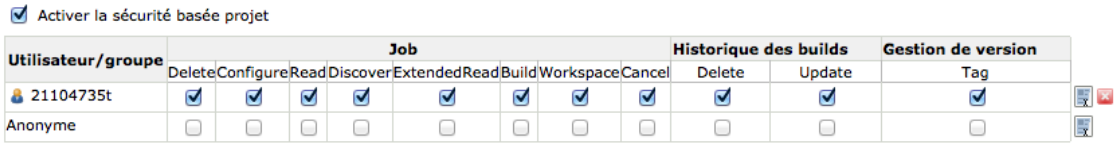

FIGURE 3.13 – Jenkins, sécurité.

2. **Configuration, étape 1 :** Vous pouvez bien évidemment configurer le projet selon votre envie, mais au début, il vaut mieux aller au plus court.

   Dans la partie gestion de code source, assure vous que le repository URL pointe vers votre dépôt SVN. Il vous sera demandé vos "credentials" à un moment : ce sont vos identifiants de l'école

(numérot étudiant/mot de passe associé).

Sur le screenshot, vous remarquerez la présence d'un @HEAD et d'une stratégie de checkout particulière : "*Toujours faire un checkout pour avoir une nouvelle copie*" (*ce qui ralentit passablement le build !*). Cela était dû à (lors de la rédaction du manuel) un déreglement de l'horloge du serveur Redmine.
Si tout se passe bien, vous n'aurez pas à rajouter @HEAD à la fin de votre URL, et vous pourrez choisir la stratégie "*Utiliser SVN update autant que possible*" qui fera alors parfaitement l'affaire.

FIGURE 3.14 – Jenkins, Configuration étape 2

3. **Configuration, étape 2 :** Vous pouvez ensuite commander à jenkins, d'éxécuter votre build **continuellement** (et c'est tout l'intérêt) !

   Ici on demande à Jenkins de construire notre build toutes les heures mais AUSSI de scruter l'outil de gestion de version toutes les minutes. Ainsi à chaque commit d'un développeur travaillant sur le projet, Jenkins va lancer un build et les tests allant avec ! Ainsi **à la moindre régression de code**, tous les développeurs seront mis au courant en consultant l'interface web de Jenkins.

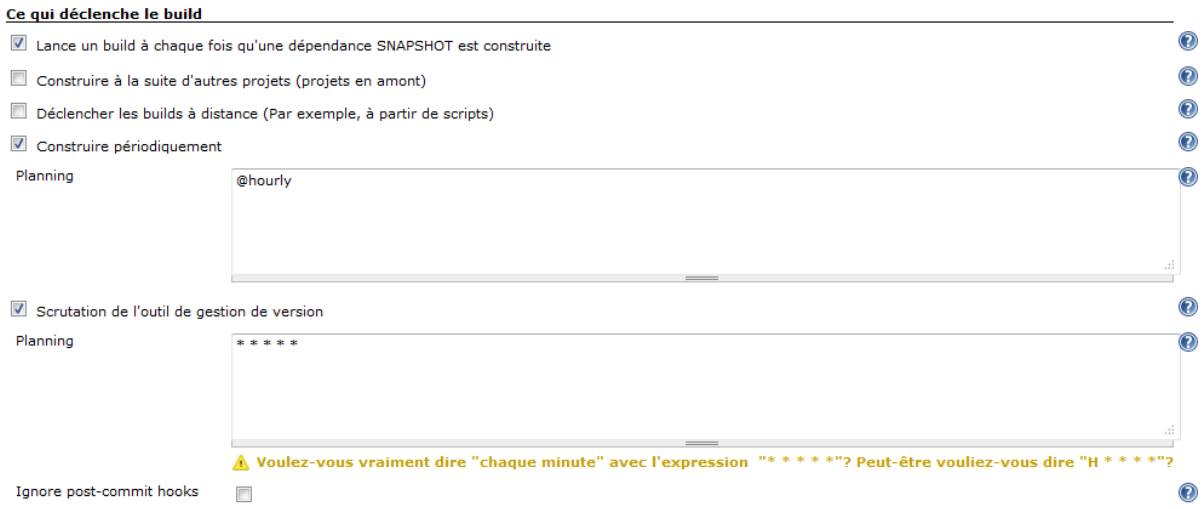

FIGURE 3.15 – Jenkins, Configuration étape 3

4. **Configuration, étape 3 :** Passons à présent à l'étape de build, qui sera décomposé en 2 parties : La génération et les tests.

   Tout d'abord ajouter une première étape au build en cliquant sur **Ajouter une étape au build**. Choisissez la version voulue d'MSBuild (v4.0 en l'occurence), et indiquez le chemin menant vers votre .sln. Ensuite, ajoutez la version de visual studio propre à vos tests (ici Visual Sudio 2010) et indiquez le chemin vers vos .dll (qui constituent pour Jenkins l'ensemble de vos tests).
   Surtout, n'oubliez pas d'indiquer un nom de fichier contenant les résultats des tests (il sera généré automatiquement !). Si besoin, vous pouvez également rajouter des options à votre ligne de commande (plus d'informations à *http ://msdn.microsoft.com/fr-fr/library/ms182489.aspx*).

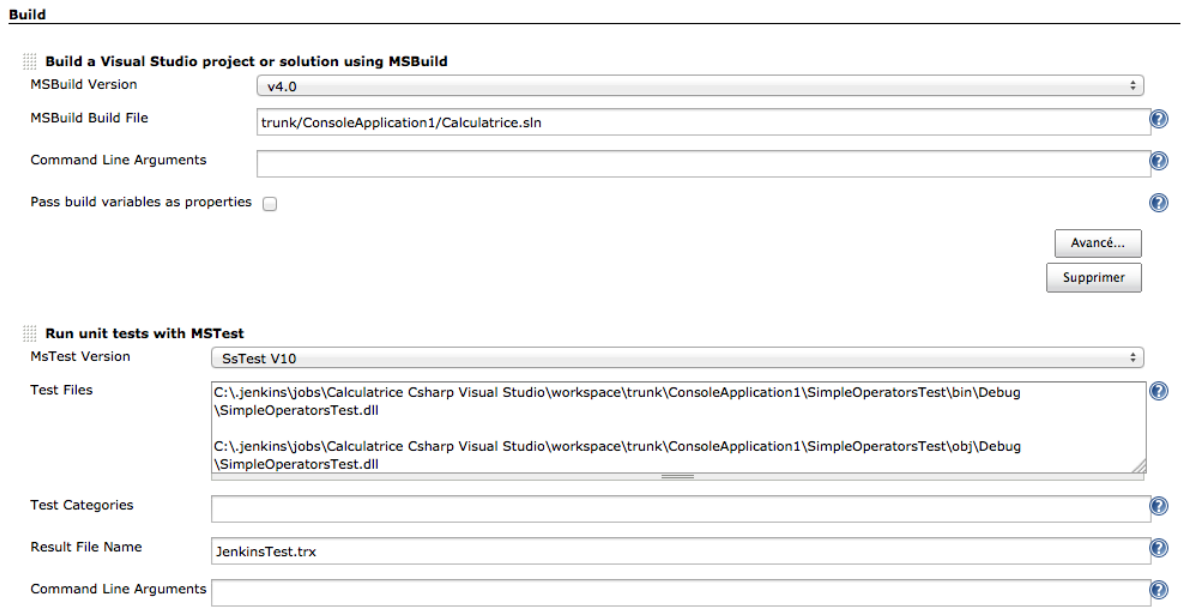

FIGURE 3.16 – Jenkins, Configuration étape 4

Si vous souhaitez avoir un affichage des résultats, à l'instar de celui proposé pour les builds Maven,

vous pouvez rajouter une étape -dite postbuild- supplémentaire en cliquant sur **ajouter une action après le build**, puis sur publish **MSTest test result report** et en indiquant le nom de votre fichier de résultats (crée durant l'étape précédente).

FIGURE 3.17 – Jenkins, Configuration étape 4

5. **Lancement du premier build :** Clickez, sur l'onglet en haut à gauche : **"Lancer un build"**. Pour être sûr et certain que tout fonctionne bien vous pouvez accèder à la sortie de la console, en cliquant sur le build en train de s'éxécuter puis sur sortie console.

FIGURE 3.18 – Jenkins, Sortie Console

Attention ! Si vous utilisez des bibliothèques qui ne sont pas incluses dans votre projet (ou que votre projet ne se suffit pas à lui même), il se peut que vous ayez quelques difficultés avec Jenkins (les bibliothèques que vous utilisez peuvent ne pas être présentes sur le serveur). Il vous faudra ainsi vous rapprochez de l'équipe du service informatique afin d'installer les bibliothèques externes nécéssaires (avec une configuration identique sur le client).

Voilà, la configuration du job Jenkins est finie ! Cependant, on va tester dans la partie suivante sa réelle efficacité...

3.0.12 Une petite vérification s'impose

Testons la réelle aptitude de Jenkins à détecter les régressions de code... *Imaginez l'un de vos coéquipiers récupérer le code source, changer le code source et de ce fait faire échouer un test qui réussissait auparavant. Ce coéquipier, sans se rendre compte de son erreur, va commiter son changement.* **Jenkins va-t-il détecter automatiquement cette régréssion ?**.

Pour cela, plusieurs étapes sont nécéssaires :

1. Mettez vous à la place de votre coéquipiez et récupèrez le code source depuis Redmine. Pour cela click droit sur votre bureau > Checkout. Indiquez l'adresse de votre dépôt SVN (comme dans la première étape).

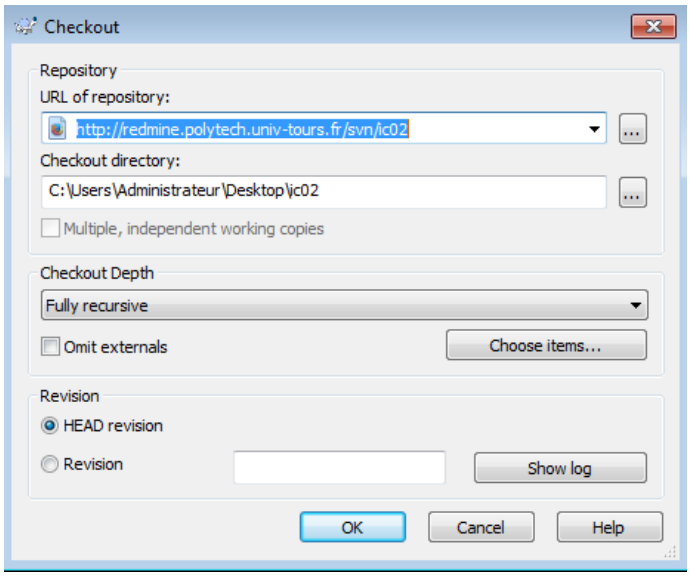

FIGURE 3.19 – Premier CheckOut

2. Comme tout à l'heure, lancez votre projet dans Visual Studio 2010 ou 2012. Modifiez l'un des tests de manière à ce que celui-ci échoue (dans mon cas j'ai modifié le test testAdd de manière à le faire échouer). Sauvegardez et fermez Visual Studio.
3. Click droit sur le dossier récupéré à la première étape, puis cliquez sur SVN Commit (afin de sauvegarder les changements..).

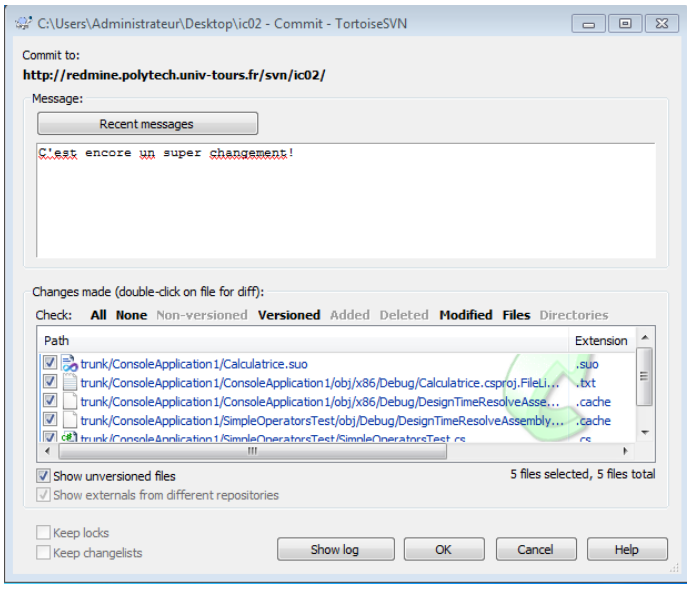

FIGURE 3.20 – C'est un super changement !

4. Enfin retournez sur votre serveur jenkins. Attendez une petite minute, et regardez votre build s'éxécuter. Si tout s'est "bien" passé, Jenkins aura remarqué le changement. L'icône du build a viré jaune, et le graphique de résultats de tests a tourné au rouge... .

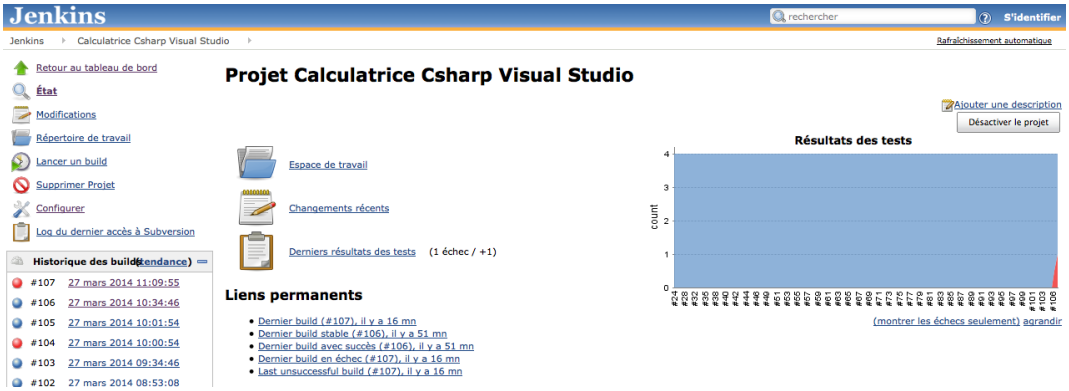

FIGURE 3.21 – Un probleme ?

En cliquant sur le build, vous aurez plus d'informations : quel test à échoué, quand a eu lieu le commit, et vous pourrez même savoir à qui est dûe cette erreur !

3.0.13 Lancement d'une analyse de qualité de code avec SonarQube

Avant tout chose, comparez l'adresse du serveur que l'on vous aura préalablement fournie avec l'adresse située dans le dossier c :/Program Files/SonarRunner/conf . Si celles-ci sont différentes, remplacez l'adresse ip 10.172.5.44(que vous pouvez voir sur le screenshot) par celle que l'on vous a fournie. Il vous faudra à nouveau vous connecter avec vos identifiants habituels (numéro étudiant suivi de t, et mot de passe associé).

```
#----- Default SonarQube server
sonar.host.url=http://10.172.5.44:8080/sonar

#----- PostgreSQL
sonar.jdbc.url=jdbc:postgresql://10.172.5.44:5432/sonar
sonar.jdbc.driverClassName=org.postgresql.Driver
```

FIGURE 3.22 – Configuration SonarRunner

Puis, rendez vous à la racine de votre projet et créez un fichier texte sonar-project.properties.

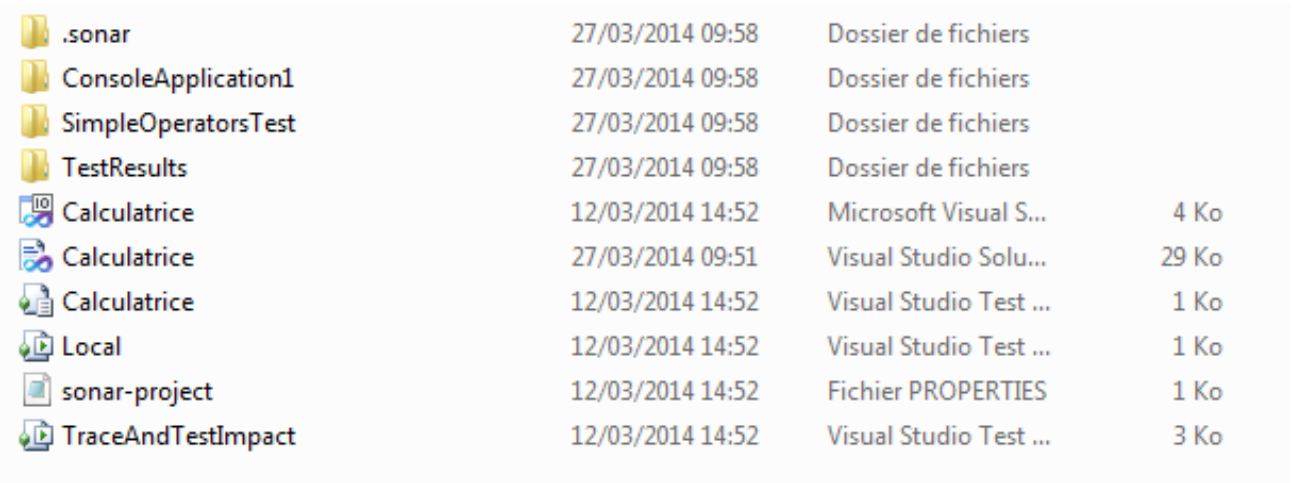

FIGURE 3.23 – Architecture du Projet

Remplissez le fichier sonar-project.properties de la manière suivante.

1. Identification du projet :
   - **sonar.projectKey=my :project**
   - **sonar.projectName=My project**
   - **sonar.projectVersion=1.0**
2. Identification des sources :
   On y indique le chemin du repertoire parent contenant les sources.
   - **sonar.sources=.**
   - **sonar.dotnet.visualstudio.solution.file=Calculatrice.sln**.
   - **sonar.dotnet.visualstudio.testProjectPattern=SimpeOperatorsTest**.
3. Précision du langage
   - **sonar.language=cs**

.

NB : Les commentaires sont précédés par le caractère#.
NB 2 : Vous pouvez trouver plus d'informations à l'adresse
http ://docs.codehaus.org/display/SONAR/Analysis+Parameters

```
# Project identification
sonar.projectKey=Sample:CalculatriceCsharp
sonar.projectVersion=1.0
sonar.projectName=C#-Calculatrice

# Info required for Sonar
sources=.
sonar.language=cs
sonar.scm.url=scm:svn:http://redmine.polytech.univ-tours.fr/svn/ic02/

# The following property is not required if the SLN file is in the same folder as the sonar-project.properties file
#sonar.dotnet.visualstudio.solution.file=Calculatrice.sln
# The following property is not required as "*.Tests" is its default value
sonar.donet.visualstudio.testProjectPattern=SimpleOperatorsTest
```

FIGURE 3.24 – Configuration du fichier sonar-project

Voilà, la configuration est finie.
On va donc pouvoir lancer une analyse à l'aide de Sonar-Runner. Pour cela rendez vous, depuis une invite de commandes, dans le dossier contenant le fichier sonar-project.properties.Lancez la commande sonar-runner, suivi de -Dsonar.login=numEtudiant -Dsonar.password=votremotdepasse.

```
Microsoft Windows [version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. Tous droits réservés.

C:\Users\Administrateur>cd "Desktop/Projet Csharp/trunk/ConsoleApplication1"

C:\Users\Administrateur\Desktop\Projet Csharp\trunk\ConsoleApplication1>sonar-ru
nner_
```

FIGURE 3.25 – Lancement de l'analyse

Si tout se passe bien, vous devriez pouvoir voir l'écran suivant :

```
14:51:01.173 INFO  - -> Delete data prior to: 2009-04-02
14:51:01.204 INFO  - -> Clean C#-Calculatrice [id=8]
14:51:01.219 INFO  - <- Clean snapshot 416
14:51:01.782 INFO  - -> Clean Calculatrice [id=106]
14:51:01.798 INFO  - -> Clean SimpleOperatorsTest [id=107]
INFO: ------------------------------------------------------------------------
INFO: EXECUTION SUCCESS
INFO: ------------------------------------------------------------------------
Total time: 1:15.273s
Final Memory: 12M/116M
INFO: ------------------------------------------------------------------------
```

FIGURE 3.26 – Réussite de l'analyse

Enfin, vous pouvez accèder à l'outil SonarQube à l'adresse de votre serveur d'intégration continue suivi par *:8080/sonar*. Cliquez sur votre projet, voici ce que vous devriez avoir obtenu.

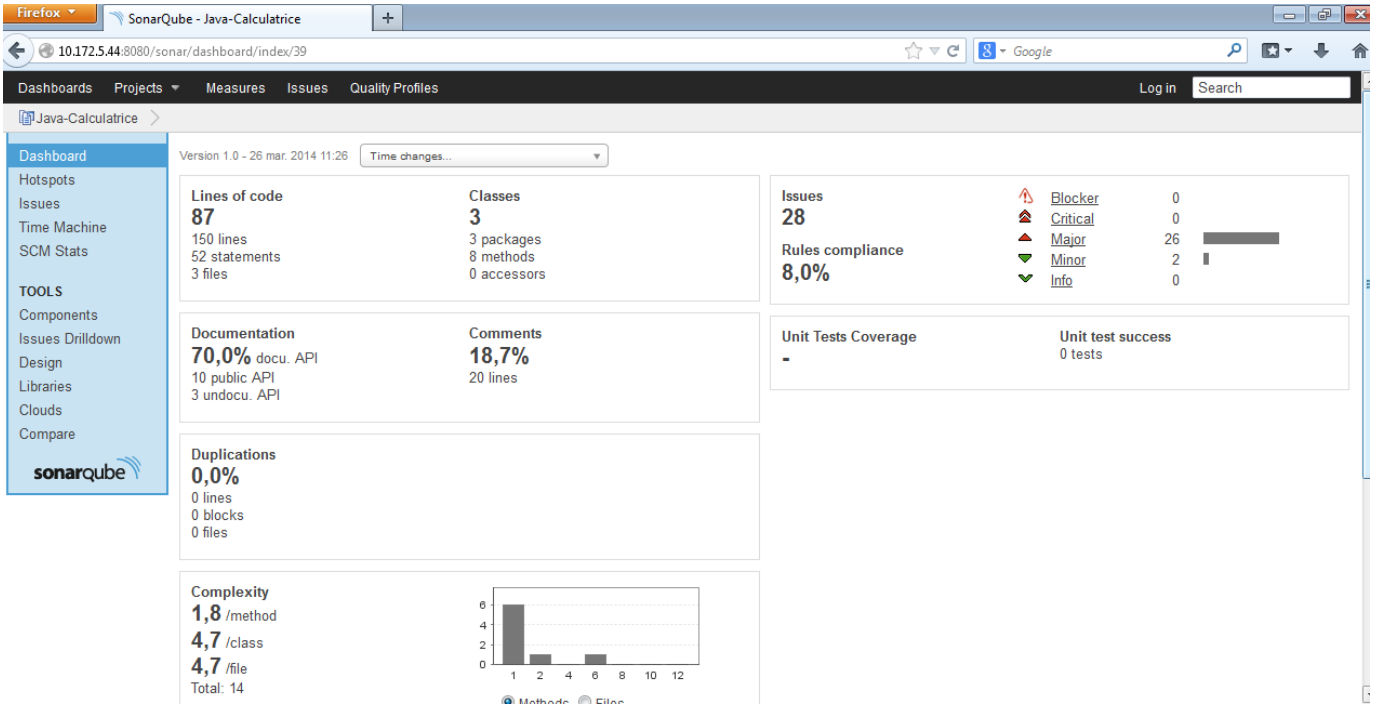

FIGURE 3.27 – Interface SonarQube

Il ne vous reste plus qu'à découvrir SonarQube... !

Projets C/C++ construits sous Visual Studio 2010 ou 2012

4.0.14 Préambule

Dans le cadre de projets C/C++ développés sous Visual Studio 2010 ou 2012, il est également conseillé d'utiliser l'une des machines clientes construites durant ce projet de fin d'études et préconfigurée pour l'utilisation du serveur d'intégration continue et qualité de code. L'une des machines virtuelles clientes contient Visual Studio 2010, l'autre Visual Studio 2012.
Si vous préferez cependant avoir votre propre machine virtuelle, vous pouvez suivre le manuel d'installation (très rapide) de la VM cliente en partant d'une machine virtuelle de base (que vous pouvez trouver sur n'importe quelle machine de l'école dans le dossier D :/).

Si vous utilisez un autre IDE que Visual Studio 2010 ou 2012, référez vous à la partie suivante "Pour les autres cas". En effet, toute cette partie va utiliser le moteur de production MSBuild (et non pas CMake ou make, ou Ant, ou tout autre moteur de production utilisé habituellement pour développer en C/C++).

Enfin vous pouvez trouver dans le dossier "Projets" fourni avec ce projet l'un des exemples (le projet c++) qui va être utilisé dans cette partie : **Projet C/C++**.

4.0.15 Fonctionnement sous l'IDE

Avant toute chose, il faut être sûr que tout marche bien du côté de votre IDE.
Pour cela plusieurs étapes sont nécéssaires :
— Lancez votre solution .sln avec l'IDE souhaité (Visual Studio 2010 ou 2012).
— Générez le projet. (Cliquez sur **Générer**, puis **Générer la solution**)
— Exécutez-le (Cliquez sur **Déboguer**, puis sur **Executez sans débogage**).

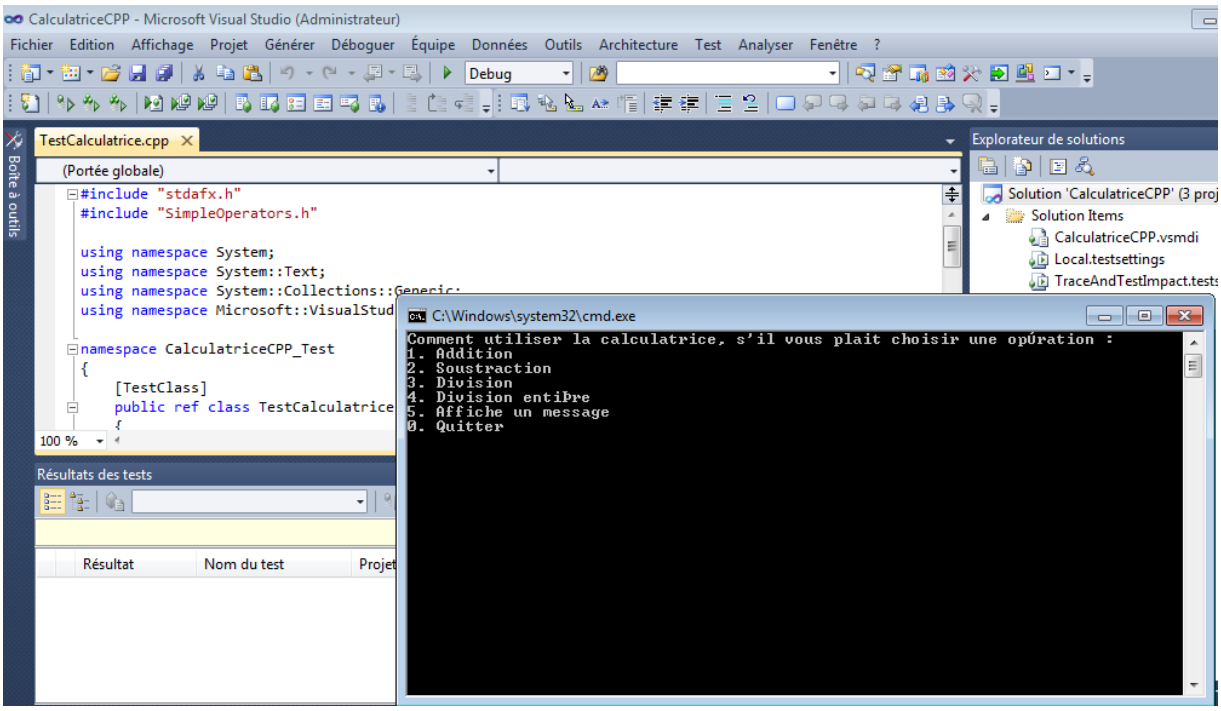

FIGURE 4.1 – Tests en ligne de commande

— Vérifiez que l'ensemble des tests s'éxécute bien (Cliquez sur **Test**, **Exécuter**, **Tous les tests de la solution**).

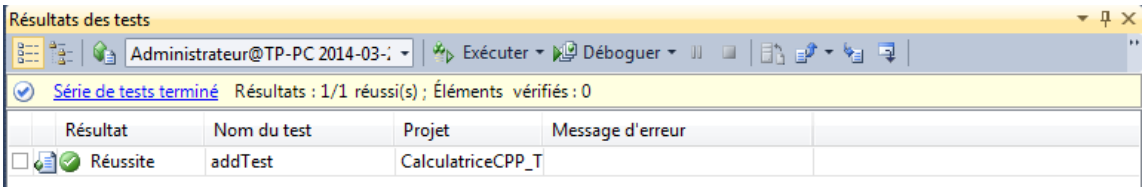

FIGURE 4.2 – Tests en ligne de commande

Voilà, votre projet est bel et bien fonctionnel sous votre IDE ! On peut donc passer à la suite et attaquer le travail sur le serveur d'intégration continue.

Attention : Une autre étape est également nécessaire. Pour avoir vos tests fonctionnant sous Jenkins, ceux-ci doivent s'éxécuter en lignes de commandes avec MsTest. Cela peut paraître anodin, mais si ce n'est pas le cas, ça ne fonctionnera pas non plus sous Jenkins.

1. Ainsi, pour vérifier le bon fonctionnement des tests en lignes de commande, allez dans le menu **Démarrer**, pointez sur **Tous les programmes**, sur **Microsoft Visual Studio 2012**, sur **Visual Studio Tools**, puis cliquez sur **Invite de commandes développeur**.
2. Pour lancer vos tests, utilisez la commande MsTest. Vous spécifierez le fichier de tests à lancer (en .dll, que vous trouverez dans le dossier Debug du projet, *dans notre cas*) à l'aide de l'option /testcontainer.

```
C:\Program Files\Microsoft Visual Studio 10.0\VC>MsTest /testcontainer:"C:/Users
/Administrateur/Desktop/Projets VS 2010/cpp/trunk/Debug/DefaultTest.dll"
Microsoft (R) Test Execution Command Line Tool Version 10.0.30319.1
Copyright (C) Microsoft Corporation 2005. Tous droits réservés.

Chargement en cours de C:\Users\Administrateur\Desktop\Projets VS 2010\cpp\trunk
\Debug\DefaultTest.dll...
Démarrage de l'exécution...

Résultats                     Tests de niveau supérieur
---------                     -------------------------
Réussite                      CalculatriceCPP_Test.TestCalculatrice.addTest
1/1 test(s) Réussite

Résumé
------
Série de tests Terminé.
  Réussite  1
  ----------
  Total     1
Fichier de résultats : C:\Program Files\Microsoft Visual Studio 10.0\VC\TestResu
lts\Administrateur_TP-PC 2014-03-27 15_18_34.trx
Paramètres de test :   Paramètres de test par défaut
```

FIGURE 4.3 – Tests en ligne de commande

3. Vous pouvez évidemment spécifiez plus d'options selon les besoins de votre projet. Vous trouverez plus d'informations sur *http ://msdn.microsoft.com/fr-fr/library/ms182489.aspx*.

Pour les projets C

Pour les projets C, l'utilisation de Visual Studio 2012 n'est pas des plus optimales mais est tout de même possible. En effet, afin de pouvoir utiliser le framework de test de visual studio 2010 (ou 2012) utilisant obligatoirement du code C++, il faut indiquer au compilateur la présence de fichiers codés en langage C et qui vont être utilisés pour les tests.

Cela se traduit par l'ajout, dans chaque fichier en-tête .h utilisé pour les tests, des directives suivantes :

```
#ifdef __cplusplus
extern "C"{
#endif
..... déclaration des fonctions .....
#ifdef __cplusplus
}
#endif
```

Ainsi, on a pour notre projet C :

```
#ifdef __cplusplus
extern "C"{
#endif

 double add(double number1, double number2);
 double sub(double number1, double number2);
 int divInt(int number1, int number2);

#ifdef __cplusplus
}
#endif
```

FIGURE 4.4 – Interface SonarQube

4.0.16 Importation sous Redmine

Afin de pouvoir utiliser le serveur subversion fourni avec le redmine de l'école, il peut être utile (mais pas nécessaire) d'instaer un logiciel client SVN. D'ailleurs, toutes les manipulations du manuel utilisateur seront faites à l'aide de ce client TortoiseSVN (déjà préinstallé si vous avez pris la VM cliente déjà préconfigurée).

En premier lieu, avant de commencer l'importation sous Redmine, vous aurez besoin d'une adresse de dépôt subversion. Pour cela allez à *http ://redmine.polytech.univ-tours.fr*, connectez vous avec vos identifiants Polytech.

Ensuite, deux cas s'offrent à vous :

1. Vous êtes en train de réaliser un projet dans le cadre de l'école et avez demandé la création d'un projet auprès du service informatique (démarche habituelle dans le cadre de PIL/PFE...). Dans ce cas, vous trouverez l'adresse de votre dépôt subversion dans l'onglet **Configuration**, onglet **Dépôts**.

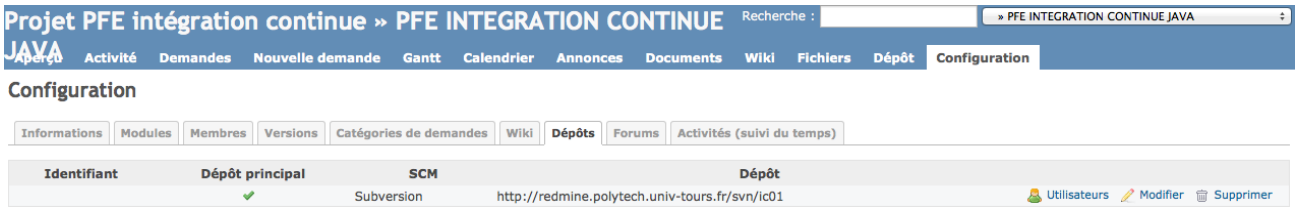

FIGURE 4.5 – Adresse du Dépôt SVN

2. Vous êtes dans le cadre d'un TP pour tester le serveur d'intégration continue. Dans ce cas-là, vous pouvez créer un projet simple (en respectant les conventions de nommage de polytech *ANNEECIVILE_PROMO_NOMPROJET* ou *ANNEECIVILE_PROMO_TP_NřETU*). Pour cela, allez dans Projets, cliquez sur Nouveau Projet.

FIGURE 4.6 – Adresse du Dépôt SVN, création d'un projet, étape 1

Puis remplissez les informations nécéssaires et demandées pour ce projet (Nom, description, identifiant...)

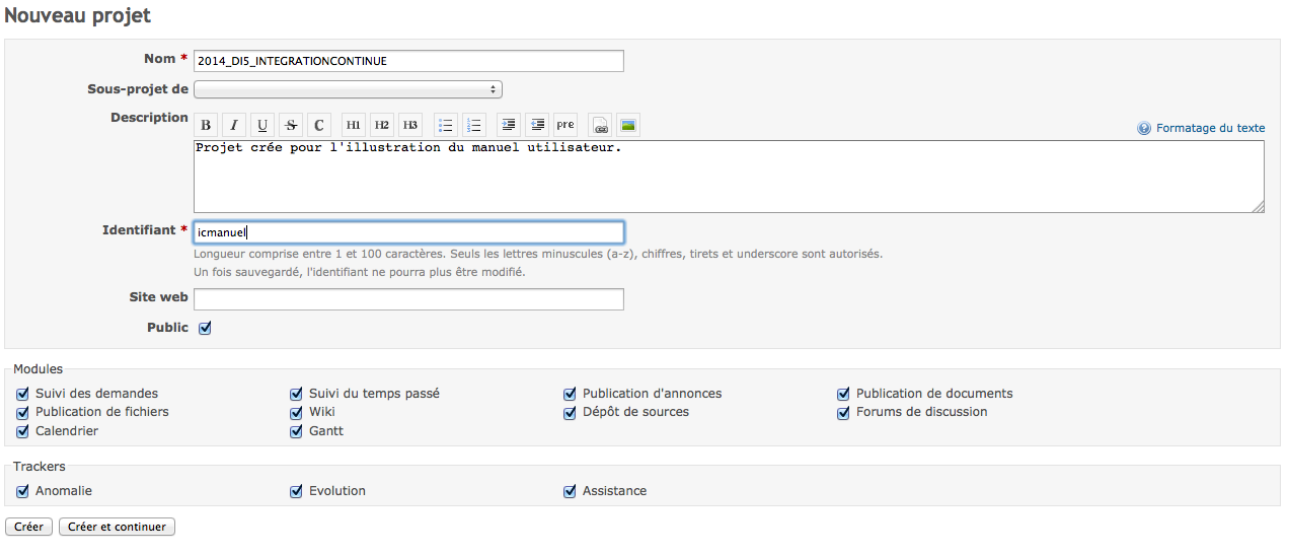

FIGURE 4.7 – Adresse du Dépôt SVN, création d'un projet, étape 2

Ensuite, vous pouvez récupérer l'adresse de votre dépôt subversion dans l'onglet **Configuration**, onglet **Dépôts**.

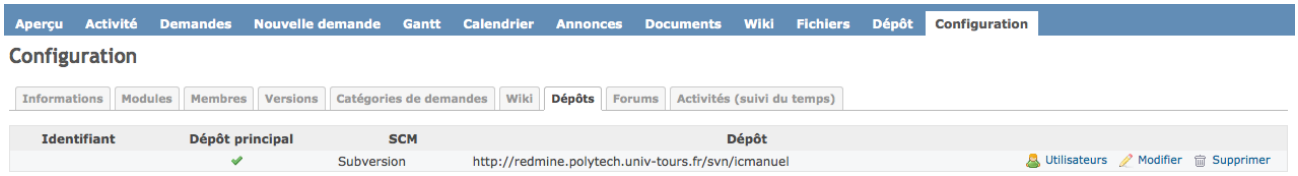

FIGURE 4.8 – Adresse du Dépôt SVN

Une fois l'adresse du dépôt subversion en votre possession, vous pouvez retourner auprès de votre projet.
Créez un dossier "NomduProjet". Dans ce dossier, créez 3 dossiers trunk, branches, tags. Placez le projet entier dans le dossier trunk.

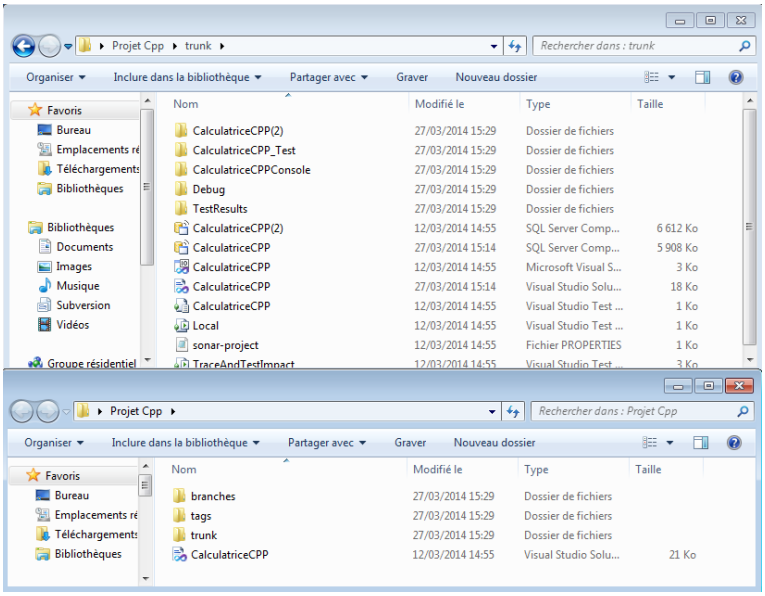

FIGURE 4.9 – Architecture des dossiers pour SVN.

Cliquez droit sur ce **projet** > **SVNTortoise** > **Import**. Rentrez l'adresse du dépôt subversion que vous venez d'obtenir, et n'oubliez pas d'ajouter un message d'importation (*c'est une bonne pratique à adopter !*).

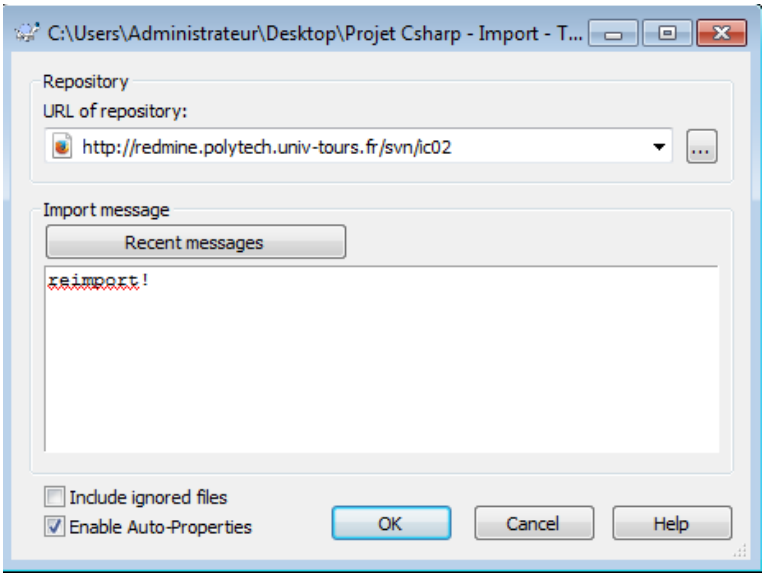

FIGURE 4.10 – Première importation.

Vous pouvez bien sûr aller vérifier sur redmine le bon fonctionnement de cette dernière opération. Allez à *http ://redmine.polytech.univ-tours.fr*, identifiez-vous si nécessaire, allez sur votre projet, onglet **Dépôt**.

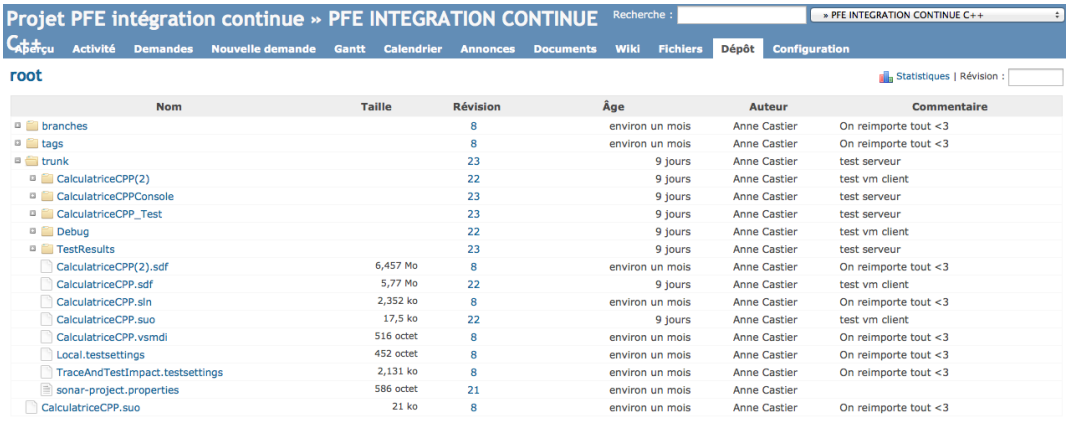

FIGURE 4.11 – Verification Importation

Ça y est, votre projet est dès à présent sur votre dépôt subversion. On va donc dès à présent pouvoir attaquer la partie plus intéressante : Jenkins.

4.0.17 Création du Job sous Jenkins

Grâce à l'adresse que l'on vous a fourni dans le cadre de ce TP, connectez vous à Jenkins à l'adresse *adresseserveur :8080/jenkins*. Il vous sera demandé de vous identifier, chose que vous pouvez faire naturellement avec vos identifiants étudiants (numéro étudiant suivi de t / password). Si vous ne disposez pas de droits d'accès supplémentaires, rendez vous auprès de l'équipe du service informatique, avec le nom de votre groupe (que vous pouvez connaître à l'adresse *adresseserveur :8080/jenkins/whoAmI*.

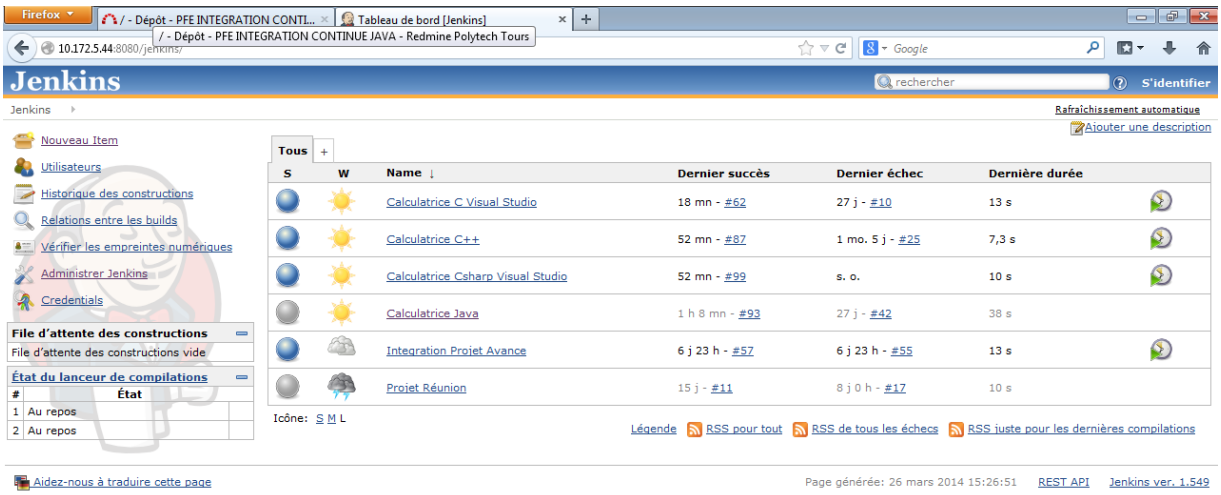

FIGURE 4.12 – Jenkins, page d'accueil

Voici les différentes étapes de configuration d'un job sous Jenkins pour un projet C# Visual Studio .

1. **Création :** Cliquez sur Nouveau Item, puis cochez l'option construire un projet freestyle et donnez un nom (sans caractères trop exotiques...) à votre job.

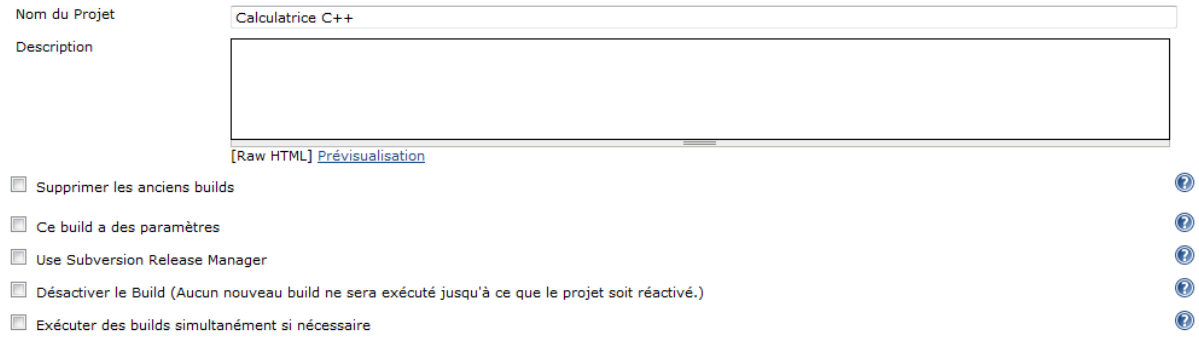

FIGURE 4.13 – Jenkins, Configuration étape 1

Bien sûr, il est fortement déconseillé de laisser l'accès à d'autres personnes. De ce fait, n'oubliez pas de vous attribuer tous les droits sur votre propre projets à vous ainsi qu'aux autres personnes/groupes impliquées dans le projet.

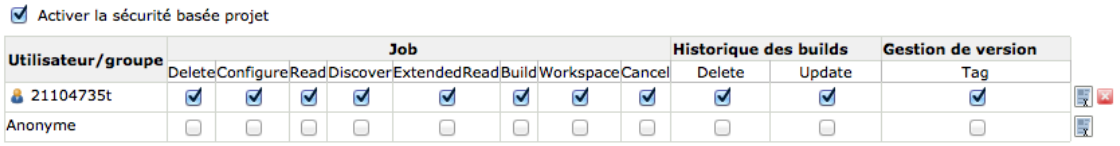

FIGURE 4.14 – Jenkins, sécurité.

2. **Configuration, étape 1 :** Vous pouvez bien évidemment configurer le projet selon votre envie, mais au début, il vaut mieux aller au plus court.

   Dans la partie gestion de code source, assure vous que le repository URL pointe vers votre dépôt SVN. Il vous sera demandé vos "credentials" à un moment : ce sont vos identifiants de l'école (numérot étudiant/mot de passe associé).

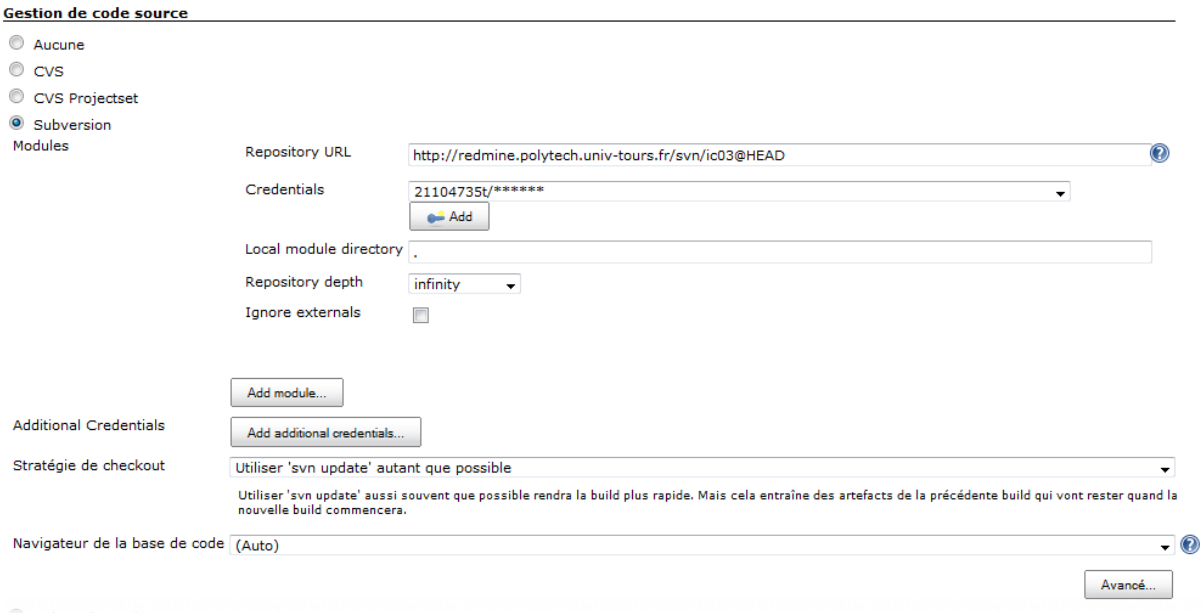

FIGURE 4.15 – Jenkins, Configuration étape 2

3. **Configuration, étape 2 :** Vous pouvez ensuite commander à jenkins, d'éxécuter votre build **continuellement** (et c'est tout l'intérêt) !

   Ici on demande à Jenkins de construire notre build toutes les heures mais AUSSI de scruter l'outil de gestion de version toutes les minutes. Ainsi à chaque commit d'un développeur travaillant sur le projet, Jenkins va lancer un build et les tests allant avec ! Ainsi **à la moindre régression de code**, tous les développeurs seront mis au courant en consultant l'interface web de Jenkins.

FIGURE 4.16 – Jenkins, Configuration étape 3

4. **Configuration, étape 3 :** Passons à présent à l'étape de build, qui sera décomposé en 2 parties : La génération et les tests.

   Tout d'abord ajouter une première étape au build en cliquant sur **Ajouter une étape au build**. Choisissez la version voulue d'MSBuild (v4.0 en l'occurence), et indiquez le chemin menant vers votre .sln. Ensuite, ajoutez la version de visual studio propre à vos tests (ici Visual Sudio 2010) et indiquez le chemin vers vos .dll (qui constituent pour Jenkins l'ensemble de vos tests).
   Surtout, n'oubliez pas d'indiquer un nom de fichier contenant les résultats des tests (il sera généré automatiquement !). Si besoin, vous pouvez également rajouter des options à votre ligne de commande (plus d'informations à *http ://msdn.microsoft.com/fr-fr/library/ms182489.aspx*).

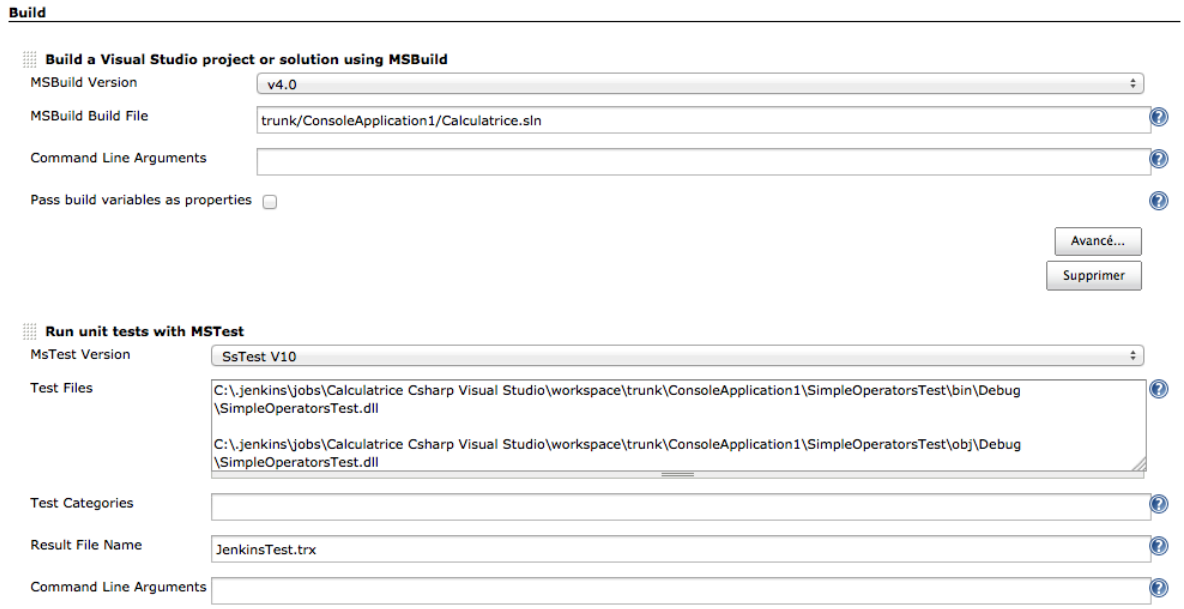

FIGURE 4.17 – Jenkins, Configuration étape 4

Si vous souhaitez avoir un affichage des résultats, à l'instar de celui proposé pour les builds Maven, vous pouvez rajouter une étape -dite postbuild- supplémentaire en cliquant sur **ajouter une action après le build**, puis sur publish **MSTest test result report** et en indiquant le nom de votre fichier de résultats (crée durant l'étape précédente).

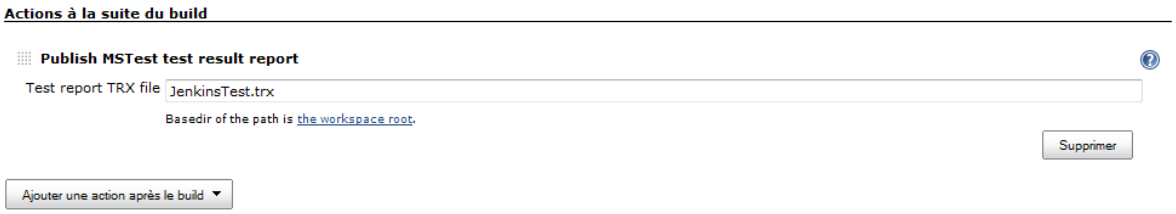

FIGURE 4.18 – Jenkins, Configuration étape 4

5. **Lancement du premier build :** Clickez, sur l'onglet en haut à gauche : **"Lancer un build"**. Pour être sûr et certain que tout fonctionne bien vous pouvez accèder à la sortie de la console, en cliquant sur le build en train de s'éxécuter puis sur sortie console.

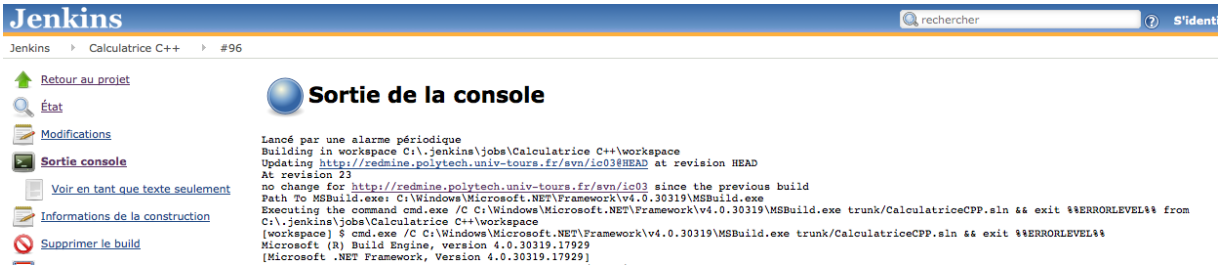

FIGURE 4.19 – Jenkins, Sortie Console

Attention ! Si vous utilisez des bibliothèques qui ne sont pas incluses dans votre projet (ou que votre projet ne se suffit pas à lui même), il se peut que vous ayez quelques difficultés avec Jenkins (les bibliothèques que vous utilisez peuvent ne pas être présentes sur le serveur). Il vous faudra ainsi vous rapprochez de l'équipe du service informatique afin d'installer les bibliothèques externes nécéssaires (avec une configuration identique sur le client).

Voilà, la configuration du job Jenkins est finie ! Cependant, on va tester dans la partie suivante sa réelle efficacité...

4.0.18 Une petite vérification s'impose

Testons la réelle aptitude de Jenkins à détecter les régressions de code... *Imaginez l'un de vos coéquipiers récupérer le code source, changer le code source et de ce fait faire échouer un test qui réussissait auparavant. Ce coéquipier, sans se rendre compte de son erreur, va commiter son changement.* **Jenkins va-t-il détecter automatiquement cette régréssion ?**.

Pour cela, plusieurs étapes sont nécéssaires :

1. Mettez vous à la place de votre coéquipiez et récupèrez le code source depuis Redmine. Pour cela click droit sur votre bureau > Checkout. Indiquez l'adresse de votre dépôt SVN (comme dans la première étape).

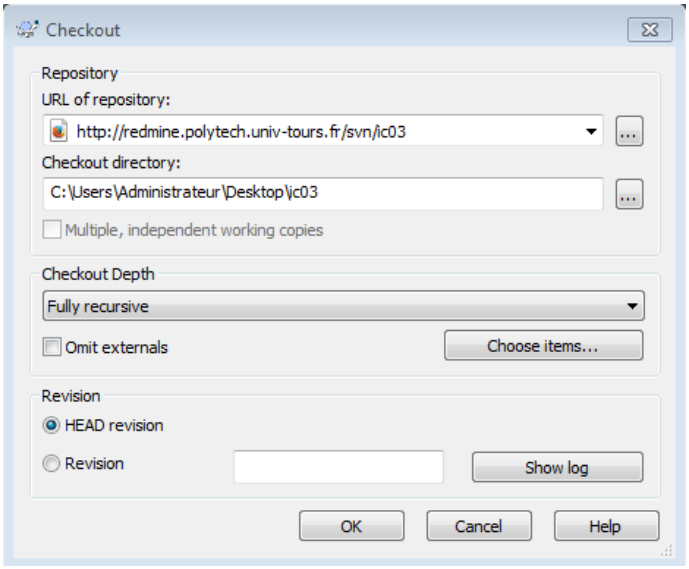

FIGURE 4.20 – Premier CheckOut

2. Comme tout à l'heure, lancez votre projet dans Visual Studio 2010 ou 2012. Modifiez l'un des tests de manière à ce que celui-ci échoue (dans mon cas j'ai modifié le test addTest de manière à le faire échouer). Sauvegardez et fermez Visual Studio.
3. Click droit sur le dossier récupéré à la première étape, puis cliquez sur SVN Commit (afin de sauvegarder les changements..).

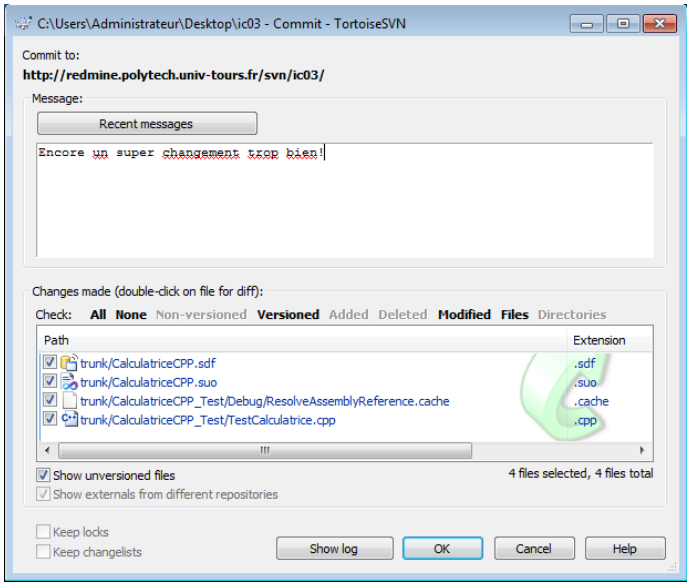

FIGURE 4.21 – C'est un super changement !

4. Enfin retournez sur votre serveur jenkins. Attendez une petite minute, et regardez votre build s'éxécuter. Si tout s'est "bien" passé, Jenkins aura remarqué le changement. L'icône du build a viré jaune, et le graphique de résultats de tests a tourné au rouge...

   En cliquant sur le build, vous aurez plus d'informations : quel test à échoué, quand a eu lieu le commit, et vous pourrez même savoir à qui est dûe cette erreur !

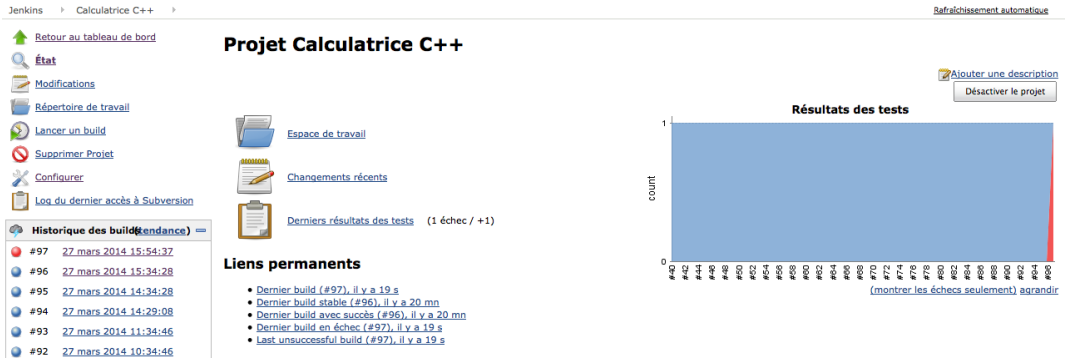

FIGURE 4.22 – Un probleme ?

4.0.19 Lancement d'une analyse de qualité de code avec SonarQube

Avant tout chose, comparez l'adresse du serveur que l'on vous aura préalablement fournie avec l'adresse située dans le dossier c :/Program Files/SonarRunner/conf . Si celles-ci sont différentes, remplacez l'adresse ip 10.172.5.44(que vous pouvez voir sur le screenshot) par celle que l'on vous a fournie. Il vous faudra à nouveau vous connecter avec vos identifiants habituels (numéro étudiant suivi de t, et mot de passe associé).

```
#----- Default SonarQube server
sonar.host.url=http://10.172.5.44:8080/sonar

#----- PostgreSQL
sonar.jdbc.url=jdbc:postgresql://10.172.5.44:5432/sonar
sonar.jdbc.driverClassName=org.postgresql.Driver
```

FIGURE 4.23 – Configuration SonarRunner

Puis, rendez vous à la racine de votre projet et créez un fichier texte sonar-project.properties.

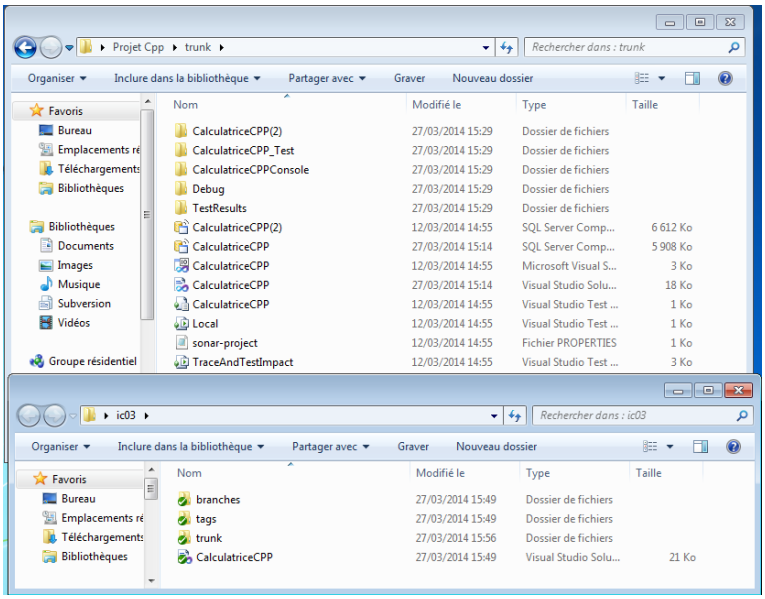

FIGURE 4.24 – Architecture du Projet

Remplissez le fichier sonar-project.properties de la manière suivante.

1. Identification du projet :
   - **sonar.projectKey=my :project**
   - **sonar.projectName=My project**
   - **sonar.projectVersion=1.0**
2. Identification des sources :
   On y indique le chemin du repertoire parent contenant les sources.
   - **sonar.sources=.**
   - **sonar.dotnet.visualstudio.solution.file=CalculatriceCPP.sln**.
   - **sonar.dotnet.visualstudio.testProjectPattern=CalculatriceCPP_Test**.
3. Précision du langage
   - **sonar.language=c++**

.

NB : Les commentaires sont précédés par le caractère#.
NB 2 : Vous pouvez trouver plus d'informations à l'adresse
http ://docs.codehaus.org/display/SONAR/Analysis+Parameters

.

```
# Project identification
sonar.projectKey=Sample:CalculatriceCPP
sonar.projectVersion=1.0
sonar.projectName=Cpp-CalculatriceCPP

# Info required for Sonar
sources=.
sonar.language=c++
sonar.scm.url=scm:svn:http://redmine.polytech.univ-tours.fr/svn/ic03/

# The following property is not required if the SLN file is in the same folder as the sonar-project.properties file
sonar.dotnet.visualstudio.solution.file=CalculatriceCPP.sln
# The following property is not required as "*.Tests" is its default value
sonar.donet.visualstudio.testProjectPattern=CalculatriceCPP_Test
```

FIGURE 4.25 – Configuration du fichier sonar-project

Voilà, la configuration est finie.
On va donc pouvoir lancer une analyse à l'aide de Sonar-Runner. Pour cela rendez vous, depuis une invite de commandes, dans le dossier contenant le fichier sonar-project.properties.Lancez la commande sonar-runner, suivi de -Dsonar.login=numEtudiant -Dsonar.password=votremotdepasse.

```
C:\Users\Administrateur\Desktop\Projets VS 2010\cpp>cd trunk

C:\Users\Administrateur\Desktop\Projets VS 2010\cpp\trunk>sonar-runner
C:\Program Files\SonarRunner
SonarQube Runner 2.3
Java 1.7.0_51 Oracle Corporation (32-bit)
```

FIGURE 4.26 – Lancement de l'analyse

Si tout se passe bien, vous devriez pouvoir voir l'écran suivant :

```
INFO: ------------------------------------------------------------------------
INFO: EXECUTION SUCCESS
INFO: ------------------------------------------------------------------------
Total time: 35.015s
Final Memory: 13M/114M
INFO: ------------------------------------------------------------------------
```

FIGURE 4.27 – Réussite de l'analyse

Enfin, vous pouvez accèder à l'outil SonarQube à l'adresse de votre serveur d'intégration continue suivi par *:8080/sonar.* Cliquez sur votre projet, voici ce que vous devriez avoir obtenu.

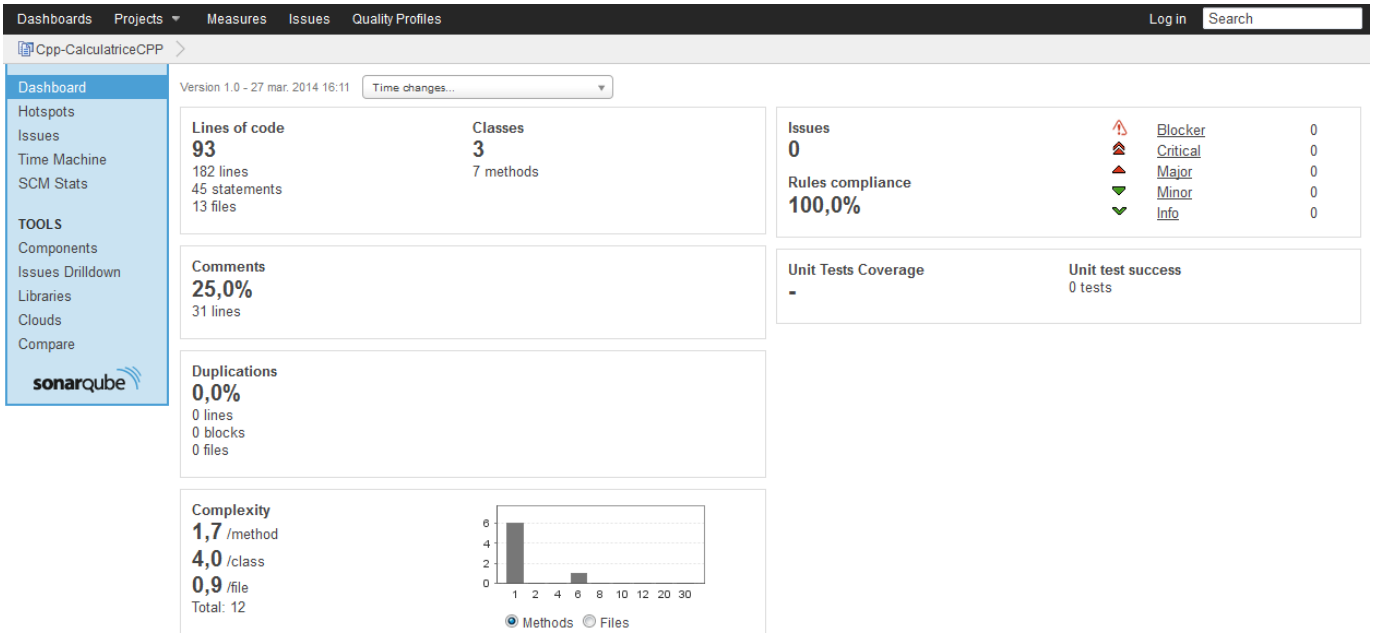

FIGURE 4.28 – Interface SonarQube

Il ne vous reste plus qu'à découvrir SonarQube... !.

Notez cependant que l'utilisation du plugin cpp (community) ne permet pas, à l'instar des plugins c# et java, de lancer une anlayse avec CppCheck. C'est donc à vous de réaliser l'analyse et fournir le rapport en .xml au fichier de configuration sonar.properties.xml pour que les vrais résultats de "Rule Compliance" soient apportés à SonarQube. Ne vous fiez donc pas au résultat de 100% de réussite à cette analyse ! Plus d'informations à : *https ://github.com/wenns/sonar-cxx*

Pour les autres cas...

Vous avez pu vous en rendre compte en parcourant les websites de Jenkins et SonarQube, que ceux-ci sont capables de prendre en charge de très nombreux langages. Pour ce PFE, nous nous sommes limitées à l'adaptation de 4 langages (C, C++, C#, Java/Maven). Cependant vous pouvez tout à fait essayer d'utiliser le serveur d'intégration continue pour d'autres projets avec différents langages (comme PHP, Ruby, Python...) ou différents moteurs de productions (CMake, Ant...) . Vous pouvez évidemment vous inspirer des précédents tutoriaux en les adaptant à vos besoins. Pour cela, il vous sera -sûrement- nécéssaire de vous rapprocher des actuels administrateurs (étudiant si poursuite du PFE ou équipe du service informatique) du serveur d'intégration continue afin que ceux-ci installent les parties nécéssaires (bibliothèques, moteurs de productions) côté serveur et les configurent sur les interfaces de Jenkins et SonarQube.

Conclusion

Grâce à ce manuel, vous avez pu prendre en main de manière rapide le serveur d'intégration continue et apprendre à utiliser deux outils reconnus : SonarQube et Jenkins. Vous pouvez donc d'ores et déjà intégrer l'utilisation du serveur pour vos PILs, PFE, et autres projets informatiques en utilisant les bonne pratiques dites d'intégration continue.Gardez cependant à l'esprit que ce manuel d'utilisation, bien sûr, n'est pas exhaustif, et vous pourrez trouver bien plus d'informations complémentaires sur la toile.

Serveur d'Intégration Continue et Qualité de Code

Département Informatique
5e année
2013 - 2014

Manuel Utilisateur

Résumé : Manuel utilisateur pour utiliser le serveur d'intégration continue et qualité de code.

Mots clefs : Manuel Utilisateur

Abstract: Operating Manual

Keywords: Operating Manual

Encadrants
Nicolas Ragot
nicolas.monmarche@univ-tours.fr
Vincent T'Kindt
tkindt@univ-tours.fr

Étudiants
Anne CASTIER
anne.castier@etu.univ-tours.fr

DI5 2013 - 2014

Université François-Rabelais, Tours

Mise en place d'un Serveur d'Intégration Continue et Qualité de Code

Département Informatique
5e année
2013 - 2014

Rapport PFE

Résumé : Rapport du projet de fin d'études de Anne Castier, encadré par Mr TKindt et Mr Nicolas Ragot, durant l'année 2013-2014 et portant sur le sujet "Mise en place d'un Serveur d'Intégration Continue et Qualité de Code"

Mots clefs : Serveur d'Intégration Continue et Qualité de Code, Projet de fin d'études.

Abstract: Report of the project of the end of studies of Anne Castier, supervised by Mr Tkindt et Mr Nicolas Ragot, during year 2013-2014 and concerning the subject "Installation of a Server of Continuous Integration and Quality Of Code

Keywords: Server of Continuous Integration and Quality Of Code, Report

---

Encadrants
Nicolas Ragot
nicolas.monmarche@univ-tours.fr
Vincent T'Kindt
tkindt@univ-tours.fr

Étudiants
Anne CASTIER
anne.castier@etu.univ-tours.fr

DI5 2013 - 2014

Université François-Rabelais, Tours